Computing and Informatics, Vol. 38, 2019, 265–290, doi: 10.31577/cai_2019_2_265

SPECIFIC PARAMETER-FREE GLOBAL OPTIMIZATION TO SPEED UP SETTING AND AVOID FACTORS INTERACTIONS

Rafael Rodríguez-Reche, Rocío P. Prado
Sebastián García-Galán, José Enrique Muñoz-Expósito
Nicolás Ruiz-Reyes

Telecommunication Engineering Department
University of Jaén
Science and Technology Campus of Linares
Jaén, Spain
e-mail: {rrreche, rperez, sgalan, jemunoz, nicolas}@ujaen.es

Abstract. Meta-heuristics utilizing numerous parameters are more complicated than meta-heuristics with a couple of parameters for various reasons. In essence, the effort expected to tune the strategy-particular parameters is far more prominent as the quantity of parameters increases and furthermore, complex algorithms are liable for the presence of further parameter interactions. Jaya meta-heuristic does not involve any strategy-specific parameters and is a one-stage technique. It has demonstrated its effectiveness compared to major types of meta-heuristics and it introduces various points of interest, such as its easy deployment and set-up in industrial applications and its low complexity to be studied. In this work, a new meta-heuristic, Enhanced Jaya (EJaya) is proposed to overcome the inconsistency of Jaya in diverse situations, introducing coherent attraction and repulsion movements and restrained intensity for flight. Comparative results of EJaya in a set of benchmark problems including statistical tests show that it is feasible to increase the accuracy, scalability and exploitation capability of Jaya while keeping its specific parameter-free feature. EJaya is especially suitable for a priori undefined characteristics optimization functions or applications where the set-up time of the optimization process is critical and parameters tuning and interactions must be avoided.

Keywords: Soft computing, optimization, meta-heuristics, parameters complexity, parameters interactions

Mathematics Subject Classification 2010: 90-08

1 INTRODUCTION

Meta-heuristics, in their genuine definition, are solution techniques that organize an association between local solution methods and more complex methodologies to make a procedure able to escape from local optima and playing out a vigorous inquiry of a solution space [8]. Over time, these techniques have incorporated new methodologies to avoid getting trapped on local optima in complex search spaces, particularly those strategies that use at least one neighbourhood structure as a method for characterizing permissible movements to change from a solution to the next, or to implement or destroy solutions in diverse applications.

Some instruments and strategies that have risen up out of research in meta-heuristic techniques have ended up being surprisingly viable, to such an extent that meta-heuristics have moved into the spotlight lately as a favoured line of assault for facing numerous sorts of complex issues, especially those of a combinatorial nature, for example, optimization in robotics [16], cloud and grid computing [6, 7], energy consumption [3], bioinformatics [18], manufacturing planning and scheduling [27], image processing [17], filter modelling [1], etc. While meta-heuristics cannot confirm the optimal character of their solutions, exact methods (which hypothetically can give such an accreditation, if permitted to run long enough) have generally demonstrated to be unable to discover solutions whose quality is near of that provided by the main meta-heuristics, especially for real-world applications, which frequently present a more complex nature. Additionally, a portion of the more effective uses of exact techniques has come to fruition with consolidating meta-heuristic methodologies inside them. These results have propelled further research and utilization of novel and enhanced meta-heuristic procedures.

It is known that there does not exist a single strategy to achieve the more efficient solution for all optimization problems, also explained as the no "free lunch" theorems for optimization [24], and since the characteristics of the problems at hand are unknown a priori in many real applications, the selection of the optimization strategy could be arbitrary. Nevertheless, even considering that the selected strategy is the most suitable for the problem in hand, generally a tuning process must be considered to adjust the value of its control parameters, what delays or even makes not feasible the consideration of optimization processes to increase the system's efficiency. All meta-heuristics are probabilistic strategies that make use of basic control parameters such as population size, number of dimensions, etc. Beyond the regular control parameters, most strategies use particular control parameters. In this way, Genetic Algorithms (GA) consider mutation, crossover and selection rates [8, 9, 28], Particle Swarm Optimization (PSO) requires the specification of inertia weight, social and cognitive controlling factors [10, 13], Artificial Bee Colony (ABC) [11, 12] must define the amount of onlooker, employed and scout bees and also, a bound factor, Harmony Search (HS) makes use of memory and pitch adjusting rates, and the amount of improvisations. Likewise, different algorithms, for example, Differential Evolution (DE), Heat Transfer Search (HTS), Biogeography-Based Optimization (BBO), Adaptive Segregational Constraint Handling Evolutionary Algorithm (AS-

CHEA), etc., require the tuning of strategy-particular parameters [8, 19, 20, 23]. The tuning of the strategy-particular parameters is an exceptionally significant aspect which highly influences the successful execution of most meta-heuristics. An inefficient tuning of strategy-particular parameters either increases the computational cost or offers local optimal solutions. Henceforth, if two meta-heuristics generate comparable results but, however, one is fundamentally less complex than the other, then the simplest is a better strategy [8]. Less complex algorithms have various points of interest, including being easy to deploy and set up in an industrial setting and being less complex to be studied.

Jaya [20] is an extremely simple algorithm, whose main virtue is given by the fact that it is not necessary to configure any control parameter to make it work beyond the own associated to the problem to be solved (e.g., delimitation of the search space, number of variables, fitness function, etc.), which makes it especially suitable for problems where characteristics are unknown a priori. Jaya efficiency has been tested in a diverse test-bed of benchmark functions [20] and the outcomes have been compared to the accomplishments of major types of algorithms, such as GA [9], PSO [13], DE [23], ABC [11] and recent simple meta-heuristics such as Teaching-Learning-Based Optimization (TLBO) [2, 21]. Results show the satisfactory performance of Jaya in a wide range of optimization problems and statistical tests additionally accredit the success of this technique. As stated before, there may not exist a better algorithm for all different types of applications, but Jaya emerges as a competitive strategy to be considered in the field of optimization. What can be stated with certainty is that Jaya is a strategy easy to implement, it requires no strategy-particular parameters and it gives the optimal solutions with slightly less time complexity and exactly the same computational complexity than major types of meta-heuristics such as the ones cited above. Thus, the optimization research community is urged to make changes to Jaya in a way that the strategy can turn out to be a great deal more accurate with a more efficient performance [20].

In this work, Jaya is redefined and the new proposal is called EJaya. EJaya overcomes Jaya inconsistency in diverse conditions through the introduction of coherent attraction and repulsion movements and restrained intensity for flight. The proposal is tested considering a wide range of benchmark problems from the Congress on Evolutionary Computation (CEC) [15]. The field of meta-heuristics, included within what is known as Evolutionary Computation, has its own space within CEC annually where the latest developments are discussed on this matter and a number of objective functions or benchmark functions to be optimized by meta-heuristics algorithms are presented as well as modifications to traditional functions in optimization problems, each with a number of features that have different impacts on the performance of meta-heuristics. Thus, benchmark functions from the CEC for the single objective real-parameter numerical optimization 2014 [15] are considered to test the enhanced strategy performance. Results including statistical tests show that it is feasible to increase the accuracy, scalability and exploitation capability of Jaya while keeping its control parameter-free feature, what makes it especially suitable for a priori undefined characteristics optimization functions or applications

where the set-up time of the optimization is critical and tuning and interactions of parameters must be avoided.

This paper is organized as follows. In Section 2, an analysis of the complexity of meta-heuristics and related works are presented. Next, the fundamentals and accomplishments of Jaya are presented in Section 3 and the proposed meta-heuristic EJaya is explained in Section 4. In Section 5, the experimental results are presented and discussed and finally, in Section 6, the main conclusions of the work are drawn.

2 BACKGROUND

Various measures of complexity exist for meta-heuristics [8]. Some measurements incorporate the quantity of phases of pseudo-code expected to depict the strategy or the quantity of lines of program expected to execute the meta-heuristic. In any case, this kind of measurements are not especially helpful, as they differ in the light of the programming language, the style and the level of the description of the pseudo-code. A more relevant measure for the complexity nature of a meta-heuristic is the quantity of parameters utilized as a part of the strategy. Parameters can be defined as the adaptable factors of a meta-heuristic that can be tuned to adjust its execution. They can be specified statically (e.g., selection rate of 0.4) or depending on the specific case (e.g., selection rate of 0.01 n, where n is the number of variables to be optimized for each individual in the population). In both of these cases, the steady estimation of the parameter or its relation to other factor in the problem in hand must be specified a priori by the strategy designer.

Most sorts of algorithms consider various specific parameters to be determined before their execution.

Table 1 presents fundamental parameters required for main types of strategies. Although these are just examples to show some typical specific parameters in various sorts of algorithms, most meta-heuristics need a diverse amount of parameters. For example, Tabu Search (TS) technique can only have one parameter, the Tabu rundown length. Nevertheless, in [26] TS in the vehicle routing issue utilizes 32 parameters. Similarly, algorithms can require less than the "base" amount of parameters by joining parameters with similar esteem. For example, the GA strategy for the spanning tree problem [25] utilizes only one specific parameter, which lets both control the population size and fix the end criteria.

Meta-heuristics utilizing numerous parameters present further complexity than the methodologies with a couple of parameters for various reasons. To begin with, the cost expected to set up and comprehend a wide set of parameters is far more prominent as the quantity of parameters increases. A brute force strategy aiming to tune m parameter values for each of the n parameters in the problem in hand, must test m^n combinations for each problem instance. Let us consider that three values can be assigned to each parameter of a strategy which requires the use of two parameters and seven parameters. In the first case, this would mean 9 evaluations and in the second case, $3^7 = 2\,187$. If this number could be considered viable, the eval-

Meta-Heuristics	**Specific Control Parameters**
Genetic Algorithms	Mutation probability
	Crossover probability
	Selection operator
Particle Swarm Optimization	Inertia weight
	Social parameter
	Cognitive parameter
Differential Evolution	Mutation rate
	Recombination rate
Artificial Bee Colony	Onlooker bees
	Employed bees
	Scout bees
	Limit
Harmony Search	Distance bandwidth
	Memory size
	Pitch adjustment rate
	Rate of choosing from memory

Table 1. Specific control parameters for major types of meta-heuristics

uations required for a 32 parameter meta-heuristic, $3^{32} = 1\,853\,020\,188\,851\,841$, are not feasible in most real-world applications. Moreover, the number of possibilities for strategies with a higher number of parameters increases exponentially, making the set-up of algorithms much harder. Despite the fact that there are approaches to search for good combinations of parameters, the number of options still increases with the quantity of parameters, which means that the set up is much more troublesome. Greater quantities of parameters additionally make the understanding of the optimization process much more difficult.

Furthermore, the complexity in setting up it is not the only drawback of complex meta-heuristics. A major problem is given by the greater possibilities of a vast parameter set to present complicated parameter interactions. Complex interactions of parameters can derive in, for example, finding numerous local solutions. In general, it is proved that the optimization of parameters independently or in small sets is not effective and the problem becomes harder as the number of parameters increases. There exist a number of works related to parameter interactions. For example, in [4] the researchers could appreciate non-trifling interactions in GA considering just three parameters. It was observed that the adequacy of a given parameter combination is frequently subject to the problem and functions to be optimized and so, it was difficult to classify and automatize the analysis of interactions. Thus, it is frequently extremely hard to keep away from parameter interactions and the level of these interactions increases drastically with the quantity of parameters once again. This has also aimed research in optimization. The recently presented TLBO [2, 20, 21] and Jaya [20] meta-heuristics do not use any strategy-particular parameters. However, it must be noted that TLBO requires two differentiated stages (i.e., educator and

learner stages), whereas Jaya considers only one stage and thus, it is more straightforward to use. Furthermore, as shown in the next section, results for Jaya in a wide range of benchmark functions provide better results than TLBO.

Hence, considering the relevance of the parameters in the performance of meta-heuristics, it is important to propose and analyse new ways of reducing the complexity of meta-heuristics while offering good solutions and this work represents a new effort in this sense.

3 FUNDAMENTALS AND ACCOMPLISHMENTS OF JAYA

Jaya could be defined as a swarm type strategy in which the attraction to the best local is eliminated and replaced by a repulsion to the worst particle in the population [20]. Moreover, the inertial weight of the particles is removed: its value is fixed to the unity and it is not modified at any time during the execution of the algorithm. In addition, particles have no memory: they do not keep a record of the best solution found, neither globally nor locally (i.e., the best position of the swarm and the best position of the particle, respectively), because the particles do not move from their position if they do not find a better solution in the next iteration. Thus, interaction with the best local position of the particle is not considered, but instead a new relationship is added: a movement of escape from the worst position within the swarm in the current iteration. On the other hand, the movement of approach to the best particle in the population is preserved. The intensity of both movements, attraction and repulsion, depends solely on the distance between the particle that makes the movement and those that affect it, that is, the best and worst positions in the swarm, respectively. Hence, there are no adjustment parameters to tune the exploration of the search space (such as c_1 and c_2 in most swarm-based strategies [14]).

Next, the algorithm Jaya is formulated formally. Consider $f(x)$ the objective function to be optimized, m the number of design factors or variables (i.e., $j = 1, 2, \ldots, m$) and n the population size (i.e., $k = 1, 2, \ldots, n$). Also, consider the best individual of the population to be the candidate solution obtaining the current optimal result of $f(x)$ (i.e., $f(x)_{best}$) and, analogously, the worst candidate to be the individual obtaining the current optimal result of $f(x)$ (i.e., $f(x)_{worst}$) in the population. If $X^t_{j,k}$ represents the value of the j^{th} factor or variable for the k^{th} solution during the t^{th} iteration, then this value is modified by following Equation (1):

$$X^{t+1}_{j,k} = X^t_{j,k} + r_{1,j,t}\left(X^t_{j,best} - |X^t_{j,k}|\right) - r_{2,j,t}\left(X^t_{j,worst} - |X^t_{j,k}|\right) \tag{1}$$

where $X^t_{j,best}$ is the value of the j^{th} factor of the best individual, $X^t_{j,worst}$, is the value of the j^{th} variable of the worst candidate and $X^{t+1}_{j,k}$ is the updated value of $X^t_{j,k}$, $r_{1,j,t}, r_{2,j,t}$ being random numbers in the range [0, 1]. The term $r_{1,j,t}\left(X^t_{j,best} - |X^t_{j,k}|\right)$ shows the leaning of the candidate to move towards the best solution or attraction factor $AF^t_{j,k}$ and the term $r_{2,j,t}(X^t_{j,worst} - |X^t_{j,k}|)$ indicates the leaning of the solution

to go away from the worst solution or rejection factor $RF^t_{j,k}$. $X^{t+1}_{j,k}$ performance is tested. All particles whose modification represents an improvement in the final results are kept up and they are considered in the following steps of the strategy. Jaya algorithm is detailed in Algorithm 1. As observed, the algorithm continuously tries to get nearer to the best solution and tries to maintain a strategic distance from the worst solution. Jaya aims to wind up successful in achieving the best solution and thus, it is named Jaya (i.e., triumph). These characteristics provide Jaya a strong convergent behaviour mainly due to its large exploration capacities and the elimination of the possibility for particles to move towards positions offering worse results than the current solution.

Algorithm 1 Jaya pseudo-code

```
-- Data
N: Number of individuals
D: Number of dimensions
-- Algorithm
Population initialization
while !end condition do
    Find (X^t_{j,best}) and worst (X^t_{j,worst}) individual in the population
    for k ¡ N do
        for j ¡ D do
            X^{t+1}_{j,k} = X^t_{j,k} + r_{1,j,t} (X^t_{j,best} - |X^t_{j,k}|) - r_{2,j,t} (X^t_{j,worst} - |X^t_{j,k}|)
        end for
        if Better solution found over particle's actual solution then
            Update particle's solution
        else
            Preserve previous particle's solution
        end if
    end for
end while
```

In [20], Jaya is evaluated in a test-bed of well-known benchmark functions in the optimization literature with diverse features like unimodality and multimodality, separability and non-separability, regularity and non-regularity, etc., where the quantity of design factors and their extents are diverse for every case. To assess the execution of the proposed algorithm based on Jaya in this work, the outcomes obtained by Jaya are contrasted to the outcomes of diverse optimization meta-heuristics, such as GA, PSO, DE, ABC and TLBO. This selection of strategies for comparison considers both the more competitive and well-known strategies (e.g., GA, PSO and DE) and the more recent and simple optimization meta-heuristics (e.g., ABC and TLBO) of those extensively analysed in [20]. Obtained mean results are reproduced in Table 2 for the diverse strategies.

From Table 2 it is observed that the results of Jaya are either equal or more accurate than the rest of the strategies in the test-bed. This can be further proved considering statistical tests. The p-value of Friedman Ranks test of these results is

f	GA	PSO	DE	ABC	TLBO	Jaya
Sphere	1.11e+03	0.00e+00	0.00e+00	0.00e+00	0.00e+00	0.00e+00
SumSquares	1.48e+02	0.00e+00	0.00e+00	0.00e+00	0.00e+00	0.00e+00
Beale	0.00e+00	0.00e+00	0.00e+00	0.00e+00	0.00e+00	0.00e+00
Easom	−1.00e+00	−1.00e+00	−1.00e+00	−1.00e+00	−1.00e+00	−1.00e+00
Matyas	0.00e+00	0.00e+00	0.00e+00	0.00e+00	0.00e+00	0.00e+00
Colville	1.49e−02	0.00e+00	4.09e−02	9.29e−02	0.00e+00	0.00e+00
Trid 6	−4.99e+01	−5.00e+01	−5.00e+01	−5.00e+01	−5.00e+01	−5.00e+01
Trid 10	1.93e−01	0.00e+00	0.00e+00	0.00e+00	0.00e+00	−2.10e+02
Zakharov	1.33e−02	0.00e+00	0.00e+00	2.47e−04	0.00e+00	0.00e+00
Schwefel 1.2	7.40e+03	0.00e+00	0.00e+00	0.00e+00	0.00e+00	0.00e+00
Rosenbrock	1.96e+05	1.50e+01	1.82e+01	8.87e−02	1.62e−05	0.00e+00
Dixon-Price	1.22e+03	6.67e−01	6.67e−01	0.00e+00	6.67e−01	0.00e+00
Foxholes	9.98e−01	9.98e−01	9.98e−01	9.98e−01	9.98e−01	9.98e−01
Branin	3.97e−01	3.97e−01	3.97e−01	3.97e−01	3.97e−01	3.97e−01
Bohachevsky 1	0.00e+00	0.00e+00	0.00e+00	0.00e+00	0.00e+00	0.00e+00
Booth	0.00e+00	0.00e+00	0.00e+00	0.00e+00	0.00e+00	0.00e+00
Michalewicz 2	−1.80e+00	−1.57e+00	−1.80e+00	−1.80e+00	−1.80e+00	−1.80e+00
Michalewicz 5	−4.64e+00	−2.49e+00	−4.68e+00	−4.68e+00	−4.67e+00	−4.68e+00
Bohachevsky 2	6.829e−02	0.00e+00	0.00e+00	0.00e+00	0.00e+00	0.00e+00
Bohachevsky 3	0.00e+00	0.00e+00	0.00e+00	0.00e+00	0.00e+00	0.00e+00
GoldStein-Price	5.87e+00	3.00e+00	3.00e+00	3.00e+00	3.00e+00	3.00e+00
Perm	3.02e−01	3.60e−02	2.40e−02	4.11e−02	6.76e−04	0.00e+00
Hartman 3	−3.86e+00	−3.63e+00	−3.86e+00	−3.86e+00	−3.86e+00	−3.86e+00
Ackley	1.46e+01	1.64e−01	0.00e+00	0.00e+00	0.00e+00	0.00e+00
Penalized 2	1.25e+02	7.67e−03	2.19e−03	0.00e+00	2.34e−08	0.00e+00
Langerman 2	−1.08e+00	−6.79e−01	−1.08e+00	−1.08e+00	−1.08e+00	−1.08e+00
Angerman 5	2.87e−01	2.13e−01	0.00e+00	2.08e−04	1.55e−05	−1.24e+00
Langerman 10	−0.63e−01	−2.566e−03	−1.05e+00	−4.46e−01	−6.49e−01	−6.20e−01
FletcherPowell 5	4.30e−03	1.45e+03	5.98e+00	1.73e−01	2.20e+00	1.59e−04
FletcherPowell 10	2.95e+01	1.36e+03	7.81e+02	8.23e+00	3.59e+01	5.43e−04

Table 2. Comparative results of Jaya with major types of meta-heuristics (mean)

1.475e−06, indicating that there are significant differences between the algorithms (considering a level of significance of p-value = 0.05). Moreover, in Table 3 the Friedman Ranks Post-Hoc test for Jaya against GA, PSO, DE, ABC and TLBO is presented for peer analysis based on data in Table 2. As shown, Jaya algorithm presents a statistically significant improvement over GA and PSO and no differences are statistically significant for DE, ABC and TLBO. However, considering that Jaya is a much simpler algorithm than DE, ABC and TLBO, it can be inferred that Jaya provides equally good results, but with the difference of its noticeable greater simplicity. Therefore, it can be said that Jaya is a better algorithm, what justifies research to reach improvements like the ones presented in this work.

Also, it is important to highlight that the competitive results of Jaya in comparison to major types of optimization algorithms are achieved in fair conditions in terms of computational effort. As known, the complexity of meta-heuristics can be measured based on two different criteria. Time and computational complexity. On the one hand, time complexity is based on the overall time taken by the different steps of the algorithm involving the generation of initial population, updating solution, etc. In short, Jaya is a swarm-type optimization algorithm directly derived from PSO in which the attraction to the best local particle is eliminated and replaced by a repulsion to the worst particle in the population, and in which the consideration of the inertial weight in the update of particles and the step for its modification is removed. Furthermore, particles have no record of the best solution found and no adjustment parameters (denoted as c_1 and c_2 in PSO) in most swarm-based strategies are considered. Hence, as a whole, the total time complexity is slightly reduced from the canonical PSO in which it is based. On the other hand, the computational complexity refers to the number of function evaluations required to achieve the final result. Also, in [20], it is shown that to test the performance of Jaya compared with the results obtained by the other optimization algorithms such as GA, PSO, DE, ABC and TLBO it is done considering the exactly same amount of function evaluations for the different meta-heuristics and the process is repeated 30 times for each algorithm and benchmark function. Thus, the consistency in the comparison in time and complexity effort is kept in the comparison of the Jaya algorithm with other meta-heuristics.

Algorithm of Study	**Comparison Algorithm**	**p-Value**
	GA	2.137193e−06
	PSO	2.527065e−03
Jaya	DE	5.230299e−01
	ABC	9.147621e−01
	TLBO	8.438326e−01

Table 3. Friedman Ranks Post-Hoc test for Jaya and major types of meta-heuristics (p-values)

4 PROPOSED META-HEURISTIC: EJAYA

From the study of Jaya, two incongruities between its philosophy and its implementation can be found. Firstly, it is an incoherence associated with the attraction and repulsion factors that govern the fundamental equation of Jaya, Equation (1), and secondly, an incoherence related to the intensity of the flight in relation to the distance to the worst particle. Considering these problems and the benefits associated with the simplicity of Jaya, two new improvements are proposed in this work. And the combination of them has resulted in the proposed strategy named EJaya.

4.1 Coherent Attraction and Repulsion Factors

In certain situations, it is possible that the design of Jaya goes against its own philosophy: particles are attracted to the worst solution of the population and repulsed of the best individual of the population. As explained in the previous section, Jaya is designed to attract particles, $X^t_{j,k}$, to the best particle found, $X^t_{j,best}$, and move them away from the worst particle found, $X^t_{j,worst}$, as indicated in Equation (1). However, derived from the analysis of its mathematical formulation, conditions may appear where the opposite behaviour, i.e., a departure from the best particle or an approach to the worst particle, may take place. As can be seen, both the attraction and repulsion factors in Equation (1) involve the absolute value of the position of the moving particle and this caused diverse incongruent situations. In case $X^t_{j,k}$¡ 0 and $|X^t_{j,k}| > |X^t_{j,best}|$, the proposed attraction factor moves the particle away from the overall best one. The same applies to the second part of the equation, in which a movement of rapprochement to the worst particle is suggested if $X^t_{j,k}$¡ 0 and $|X^t_{j,k}| > |X^t_{j,worst}|$.

In Table 4, we have presented four examples (i.e., cases A-D) of different situations where Jaya is wrong in its formulation to achieve its goal. Specifically, a particle X^t_k focusing in dimension $j = 5$ is considered in all cases for simplicity ($X^t_{5,k}$). In case A, the studied conditions are $X^t_{j,k}$¡ 0, $|X^t_{j,k}| > |X^t_{j,best}|$ and $X^t_{j,best} < 0$ when $j = 5$. In this case it can be deduced that being $X^t_{5,k} = -9$ and $X^t_{5,best} = -6$, the attraction factor or Jaya term inducing an approach to the best particle, $AF^t_{5,k} = +r_{1,j,t}\left(X^t_{5,best} - |X^t_{5,k}|\right)$, should give a positive value to let an approach of $X^t_{5,k}$ to $X^t_{5,best}$.

However, the attraction factor given by Jaya provides a negative attraction factor, $AF^t_{5,k} = -15 \cdot r_{1,j,t}$, which would result in a departure from the best particle when it should be an approach to it. On the other hand, case B corresponds to the situation where $X^t_{j,k}$¡ 0, $|X^t_{j,k}| > |X^t_{j,worst}|$ and $X^t_{j,worst} < 0$ when $j = 5$. Considering $X^t_{5,k} = -9$ and $X^t_{5,worst} = -4$ the factor inducing the particle to move away from the worst particle or repulsion factor, $RF^t_{5,k} = -r_{2,j,t}\left(X^t_{5,worst} - |X^t_{5,k}|\right)$, should provide a positive value to increase the distance. However, as can be seen, a positive value for the repulsion factor is obtained, $RF^t_{5,k} = 13 \cdot r_{2,j,t} > 0$, leading to an approach to the worst particle $X^t_{j,worst}$. Case C corresponds to the situation where $X^t_{j,k}$¡ 0, $|X^t_{j,k}| > |X^t_{j,best}|$ and $X^t_{j,best} > 0$ when $j = 5$. In this case $X^t_{5,k} = -9$ and $X^t_{5,best} = 5$, therefore, in this case the attraction factor AF should be positive in order to attract particle $X^t_{5,k}$ to $X^t_{5,best}$ but instead $AF^t_{5,k} = -4 \cdot r_{1,j,t} < 0$. Finally, an example of the situation presented when $X^t_{j,k}$¡ 0, $|X^t_{j,k}| > |X^t_{j,worst}|$ and $X^t_{j,worst} > 0$ for $j = 5$, is shown in case D. In this case, the RF should be negative to achieve a repulsion of $X^t_{5,k}$ from $X^t_{5,worst}$. However, once again, the original algorithm in this situation would get the opposite effect, i.e., an approach to the worst particle, since $RF^t_{5,k} = 6 \cdot r_{2,j,t} > 0$.

Hence, from these examples it can be understood that although Jaya is designed to approach particles to the best current solution and escape from the worst existing solution in the population, the equation of motion that governs Jaya does not always

$$X_k^t = [2\ 7\ 4\ 8\ -9\ 1\ 3\ 4\ 6] \Rightarrow X_{5,k}^t = -9$$

Case A: $X_{j,k}^t < 0, |X_{j,k}^t| > |X_{j,best}^t|, X_{j,best}^t < 0$

$$X_{best}^t = [8\ 1\ 3\ 4\ -6\ 7\ 2\ 3\ 1] \Rightarrow X_{5,best}^t = -6$$

$$X_{5,best}^t - |X_{5,k}^t| = -6 - 9 = -15$$

$$AF_{5,k}^t = +r_{1,j,t}\left(X_{5,best}^t - |X_{5,k}^t|\right) = -15 \cdot r_{1,j,t};\ r_{1,j,t} \in [0,1]$$

$$AF_{5,k}^t = -15 \cdot r_{1,j,t} < 0$$

$$\Rightarrow AF_{5,k}^t \textit{ should be } > 0 \textit{ to attract } X_{5,k}^{t+1} \textit{ to } X_{5,best}^t$$

Case B: $X_{j,k}^t < 0,\ |X_{j,k}^t| > |X_{j,worst}^t|,\ X_{j,worst}^t < 0$

$$X_{worst}^t = [2\ 5\ 7\ 2\ -4\ 9\ 8\ 5\ 9] \Rightarrow X_{5,worst}^t = -4$$

$$X_{5,worst}^t - |X_{5,k}^t| = -4 - 9 = -13$$

$$RF_{5,k}^t = -r_{2,j,t}\left(X_{5,worst}^t - |X_{5,k}^t|\right) = 13 \cdot r_{2,j,t};\ r_{2,j,t} \in [0,1]$$

$$RF_{5,k}^t = 13 \cdot r_{2,j,t} > 0$$

$$\Rightarrow RF_{5,k}^t \textit{ should be } < 0 \textit{ to move away } X_{5,k}^{t+1} \textit{ from } X_{5,worst}^t$$

Case C: $X_{j,k}^t < 0,\ |X_{j,k}^t| > |X_{j,best}^t|,\ X_{j,best}^t > 0$

$$X_{best}^t = [8\ 1\ 3\ 4\ 5\ 7\ 2\ 3\ 1] \Rightarrow X_{5,best}^t = 5$$

$$X_{5,best}^t - |X_{5,k}^t| = 5 - 9 = -4$$

$$AF_{5,k}^t = +r_{1,j,t}\left(X_{5,best}^t - |X_{5,k}^t|\right) = -4 \cdot r_{1,j,t};\ r_{1,j,t} \in [0,1]$$

$$AF_{5,k}^t = -4 \cdot r_{1,j,t} < 0$$

$$\Rightarrow AF_{5,k}^t \textit{ should be } > 0 \textit{ to attract } X_{5,k}^{t+1} \textit{ to } X_{5,best}^t$$

Case D: $X_{j,k}^t < 0,\ |X_{j,k}^t| > |,\ X_{j,worst}^t > 0$

$$X_{worst}^t = [2\ 5\ 7\ 2\ 3\ 9\ 8\ 5\ 9] \Rightarrow X_{5,worst}^t = 3$$

$$X_{5,worst}^t - |X_{5,k}^t| = 3 - 9 = -6$$

$$RF_{5,k}^t = -r_{2,j,t}\left(X_{5,worst}^t - |X_{5,k}^t|\right) = 6 \cdot r_{2,j,t};\ r_{2,j,t} \in [0,1]$$

$$RF_{5,k}^t = 6 \cdot r_{2,j,t} > 0$$

$$\Rightarrow RF_{5,k}^t \textit{ should be } < 0 \textit{ to move away } X_{5,k}^{t+1} \textit{ from } X_{5,worst}^t$$

Table 4. Incoherent cases in Jaya (A-D)

meet the premise of the algorithm. According to the sign and relative position of each particle to the best and worst particles in the population, situations occur in which the particles go far away from the leader and closer to the worst individual, as studied in the previous examples. Alternatively, in this paper we have proposed to eliminate the absolute values of the equation of motion of Jaya, formulating it in the following terms:

$$\begin{gathered} |X_{j,k}^t| \to X_{j,k}^t, \\ X_{j,k}^{t+1} = X_{j,k}^t + r_{1,j,t}\left(X_{j,best}^t - X_{j,k}^t\right) - r_{2,j,t}(X_{j,worst}^t - X_{j,k}^t). \end{gathered} \tag{2}$$

This change ensures that regardless of $X_{j,k}^t$ is greater or lower than 0, $X_{j,best}^t$ or $X_{j,worst}^t$ are greater or lower than 0 and $|X_{j,k}^t|$ is greater or lower than $|X_{j,best}^t|$ or $|X_{j,worst}^t|$, a coherent attraction $AF_{j,k}^t$ or coherent repulsion $RF_{j,k}^t$ is obtained. In Table 5, the new values obtained considering the adopted solution in the cases A-D (Table 4) are presented. As can be seen, in all the cases the proposed amendment manages to offer a coherent response to the value of the particles. Thus, this first variant, Convergent Jaya (CJaya), allows to enhance the convergence of the original algorithm slightly sacrificing its ability to explore.

4.2 Restrained Intensity for Flight

If Equation (2) is analysed, it could be observed that the flight movement, governed by $|X_{j,worst}^t - X_{j,k}^t|$, wins intensity when the distance is longer to the worst particle, which gives rise to contradictory situations: when a particle is close to the overall worst, it moves away from it very slowly, while if it is far from it, it will move away very quickly. In addition, the first proposed variant of Jaya (CJaya), can still lead to situations in which each individual can see its approach to the best particle, whose intensity is governed by $|X_{j,best}^t - X_{j,k}^t|$, diminished by the presence of the worst particle. This problem can be analysed considering the example presented in Table 6, case E. In case E, it can be observed that the escape movement of particle $X_{5,k}^t = -900$ from the worst particle $X_{5,worst}^t = 40\,000$ is more intense than the attraction movement of the particle $X_{5,k}^t = -900$ to the best $X_{5,best}^t = -6$, although it is much farther away from the worst particle. That is, when $|X_{j,best}^t - X_{j,k}^t| < |X_{j,worst}^t - X_{j,k}^t|$ (in this particular case 894 ¡ 40900) the flight movement overshadows the approach towards the best particle. In case E, it is shown that after the proposed movements of approach and flight, the resulting particle, $X_{5,k}^{t+1} = -20\,012$, is farther from the best particle $X_{5,best}^t = -6$ than before carrying out such operations. As can be inferred, it is incoherent that moving a particle away from a bad distant individual can harm its approach to a good nearby solution.

Hence, this work also proposes to perform a moderate flight movement that takes into account a balance between attraction and repulsion between particles. The second variant of the proposal (EJaya) raises the same equation of motion that the first variant (CJaya), Equation (2), but it performs an additional checking of the movement of the particles. Specifically, to prevent the approach to the best

$$X_k^t = [2\ 7\ 4\ 8\ -9\ 1\ 3\ 4\ 6] \Rightarrow X_{5,k}^t = -9$$

Solved case A: $X_{j,k}^t < 0,\ |X_{j,k}^t| > |X_{j,best}^t|,\ X_{j,best}^t < 0$

$$X_{best}^t = [8\ 1\ 3\ 4\ -6\ 7\ 2\ 3\ 1] \Rightarrow X_{5,best}^t = -6$$

$$X_{5,best}^t - X_{5,k}^t = -6 + 9 = 3$$

$$AF_{5,k}^t = +r_{1,j,t}\left(X_{5,best}^t - X_{5,k}^t\right) = 3 \cdot r_{1,j,t};\ r_{1,j,t} \in [0,1]$$

$$AF_{5,k}^t = 3 \cdot r_{1,j,t} > 0$$

$$\Rightarrow AF_{5,k}^t \textit{ attracts } X_{5,k}^{t+1} \textit{ to } X_{5,best}^t$$

Solved case B: $X_{j,k}^t < 0,\ |X_{j,k}^t| > |X_{j,worst}^t|,\ X_{j,worst}^t < 0$

$$X_{worst}^t = [2\ 5\ 7\ 2\ -4\ 9\ 8\ 5\ 9] \Rightarrow X_{5,worst}^t = -4$$

$$X_{5,worst}^t - X_{5,k}^t = -4 + 9 = 5$$

$$RF_{5,k}^t = -r_{2,j,t}\left(X_{5,worst}^t - X_{5,k}^t\right) = -5 \cdot r_{2,j,t};\ r_{2,j,t} \in [0,1]$$

$$RF_{5,k}^t = -5 \cdot r_{2,j,t} < 0$$

$$\Rightarrow RF_{5,k}^t \textit{ moves away } X_{5,k}^{t+1} \textit{ from } X_{5,worst}^t$$

Solved case C: $X_{j,k}^t < 0,\ |X_{j,k}^t| > |X_{j,best}^t|,\ X_{j,best}^t > 0$

$$X_{best}^t = [8\ 1\ 3\ 4\ 5\ 7\ 2\ 3\ 1] \Rightarrow X_{5,best}^t = 5$$

$$X_{5,best}^t - X_{5,k}^t = 5 + 9 = 14$$

$$AF_{5,k}^t = +r_{1,j,t}\left(X_{5,best}^t - X_{5,k}^t\right) = 14 \cdot r_{1,j,t};\ r_{1,j,t} \in [0,1]$$

$$AF_{5,k}^t = 14 \cdot r_{1,j,t} > 0$$

$$\Rightarrow AF_{5,k}^t \textit{ attracts } X_{5,k}^{t+1} \textit{ to } X_{5,best}^t$$

Solved case D: $X_{j,k}^t < 0,\ |X_{j,k}^t| > |,\ X_{j,worst}^t > 0$

$$X_{worst}^t = [2\ 5\ 7\ 2\ 3\ 9\ 8\ 5\ 9] \Rightarrow X_{5,worst}^t = 3$$

$$X_{5,worst}^t - X_{5,k}^t = 3 + 9 = 12$$

$$RF_{5,k}^t = -r_{2,j,t}\left(X_{5,worst}^t - X_{5,k}^t\right) = -12 \cdot r_{2,j,t};\ r_{2,j,t} \in [0,1]$$

$$RF_{5,k}^t = -12 \cdot r_{2,j,t} > 0$$

$$\Rightarrow RF_{5,k}^t \textit{ moves away } X_{5,k}^{t+1} \textit{ from } X_{5,worst}^t$$

Table 5. Incoherent cases in Jaya solved with the first improvement of EJaya (CJaya)

$$X_k^t = [2\ 7\ 4\ 8\ -900\ 1\ 3\ 4\ 6] \Rightarrow X_{5,k}^t = -900$$
$$X_{best}^t = [8\ 1\ 3\ 4\ -6\ 7\ 2\ 3\ 1] \Rightarrow X_{5,best}^t = -6$$
$$X_{worst}^t = [2\ 5\ 7\ 2\ 40\,000\ 9\ 8\ 5\ 9] \Rightarrow X_{5,worst}^t = 40\,000$$
$$r_{1,j,t} = r_{2,j,t} = 0.5$$

Case E: $|X_{j,best}^t - X_{j,k}^t| < |X_{j,worst}^t - X_{j,k}^t|$

$$X_{5,best}^t - X_{5,k}^t = -6 + 900 = 894$$
$$X_{5,worst}^t - X_{5,k}^t = 40\,000 + 900 = 40\,900$$
$$X_{5,k}^{t+1} = X_{5,k}^t + r_{1,j,t}(X_{j,best}^t - X_{5,k}^t) - r_{2,j,t}(X_{j,worst}^t - X_{5,k}^t)$$
$$= -9 + r_{1,j,t} \cdot 894 - r_{2,j,t} \cdot 40\,900 = -20\,012$$
$$\Rightarrow X_{5,k}^{t+1} \textit{ moves away from } X_{5,best}^t$$

Table 6. Incoherent cases in Jaya (E)

particle being eroded by the distance to the worst particle, a check of the movements of attraction and repulsion for each dimension is included, in a way that if the movement of repulsion is greater than the movement of attraction, the movement of repulsion is halved in consecutive iterations (bisection technique or binary-search method) until its magnitude is lower than the movement of attraction, avoiding oscillations of particles in problems with numerous dimensions. Analytically:

$$\textit{while} \quad |X_{j,best}^t - X_{j,k}^t| < |X_{j,worst}^t - X_{j,k}^t|$$
$$|X_{j,worst}^t - X_{j,k}^t| = \frac{|X_{j,worst}^t - X_{j,k}^t|}{2}$$
$$\textit{end}$$
$$X_{j,k}^{t+1} = X_{j,k}^t + r_{1,j,t}\left(X_{j,best}^t - X_{j,k}^t\right) - r_{2,j,t}\left(X_{j,worst}^t - X_{j,k}^t\right)$$

In Table 7, the results of applying EJaya for case E are presented.

It can be observed that when the movement of flight is moderated in relation to the movement towards the best particle, a departure from the worst particle, $X_{5,worst}^t = 40\,000$, can be achieved simultaneously to an approximation to the best one, $X_{5,best}^t = -6$, being the resulting solution $X_{5,k}^{t+1} \simeq 118.4687$ instead of $X_{5,k}^{t+1} = -20\,012$, which was obtained by only applying the first improvement of Jaya, CJaya. The pseudocode of EJaya is shown in Algorithm 2.

$$X_k^t = [2\ 7\ 4\ 8\ -900\ 1\ 3\ 4\ 6] \Rightarrow X_{5,k}^t = -900$$
$$X_{best}^t = [8\ 1\ 3\ 4\ -6\ 7\ 2\ 3\ 1] \Rightarrow X_{5,best}^t = -6$$
$$X_{worst}^t = [2\ 5\ 7\ 2\ 40\,000\ 9\ 8\ 5\ 9] \Rightarrow X_{5,worst}^t = 40\,000$$
$$r_{1,j,t} = r_{2,j,t} = 0.5$$

Solved case E: $|X_{j,best}^t - X_{j,k}^t| < |X_{j,worst}^t - X_{j,k}^t|$

$$X_{5,best}^t - X_{5,k}^t = -6 + 900 = 894$$
$$X_{5,worst}^t - X_{5,k}^t = 40\,000 + 900 = 40\,900$$

$$\textit{while } |X_{5,best}^t - X_{5,k}^t| < |X_{5,worst}^t - X_{5,k}^t| \Rightarrow$$
$$|X_{j,worst}^t - X_{j,k}^t| = \tfrac{|X_{j,worst}^t - X_{j,k}^t|}{2}$$
$$\textit{end}$$
$$|X_{j,worst}^t - X_{j,k}^t| = \tfrac{40\,900}{2^6} = 639.0625$$

$$X_{5,k}^{t+1} = X_{5,k}^t + r_{1,j,t}(X_{j,best}^t - X_{5,k}^t) - r_{2,j,t}(X_{j,worst}^t - X_{5,k}^t)$$
$$= -9 + 0.5 \cdot 894 - 0.5 \cdot 639.0625 \simeq 118.4687$$
$$\Rightarrow X_{5,k}^{t+1} \textit{ is attracted to } X_{5,best}^t$$

Table 7. Incoherent case in Jaya solved with EJaya

5 EXPERIMENTAL EVALUATION AND DISCUSSION

To make an extensive comparison of the proposed improvements to Jaya, experiments of global optimization with functions of various kinds, including low, medium and high number of dimensions D (10, 50, 100), unimodal and multimodal functions, with and without random components and with a search space with restricted and unrestricted areas have been conducted in a way that the reader is provided with a wide range of data to distinguish the strengths and weaknesses of the proposal. Some functions are retrieved from the classic literature on global optimization (unimodal functions) and most of them have been presented in CEC 2014 (multimodal functions) [15]. The characteristics that make up each function are described briefly in Table 8. For each meta-heuristic to be examined, the objective functions are tested considering 40 runs. In each run, 50 random initial solutions are generated and meta-heuristics can evaluate possible solutions up to 2 000 per number of dimension of the objective function times. The set of 40 runs is called an experiment and each experiment is configured to evaluate the objective functions considering 10, 50 and 100 dimensions or variables. For each dimension, solutions in the range

Algorithm 2 EJaya pseudocode

```
—— Data
N: Number of individuals
D: Number of dimensions
—— Algorithm
Population initialization
while !end condition do
    Find (X^t_{j,best}) and worst (X^t_{j,worst}) individual in the population
    for k ¡ N do
        for j ¡ D do
            while |X^t_{j,best} − X^t_{j,k}| < |X^t_{j,worst} − X^t_{j,k}| do
                |X^t_{j,worst} − X^t_{j,k}| = (|X^t_{j,worst} − X^t_{j,k}|)/2
            end while
            X^{t+1}_{j,k} = X^t_{j,k} + r_{1,j,t} (X^t_{j,best} − X^t_{j,k}) − r_{2,j,t} (X^t_{j,worst} − X^t_{j,k})
        end for
        if Better solution found over particle's actual solution then
            Update particle's solution
        else
            Preserve previous particle's solution
        end if
    end for
end while
```

$[-100, 100]$ are accepted unless the objective function specifies a different range, as indicated in Table 8.

This section presents the results obtained by each of the algorithms differentiating the evaluation of unimodal and multimodal functions and finally, conducting a global analysis of all of them. Results for 10, 50 and 100 dimensions (Tables 9, 10 and 11, respectively) are listed by objective functions (rows) and each function contains four sub-rows indicating the average of the found solutions, the best solution reached, the average runtime and the classification of those heuristics depending on the quality of the average reached solution. Finally, a statistical analysis of the results is presented.

Unimodal Functions Analysis

In Tables 9, 10 and 11 it can be observed that the most appropriate heuristic for solving benchmark unimodal functions (f_{01}, f_{02}, f_{03}) is EJaya, which obtains for all dimensions D (10, 50 and 100) results at least an order of magnitude better than its competitor in second place, CJaya. Unimodal functions are simple functions in which algorithms with high qualities for exploitation render exceptionally. Thus, these results are an indication of the high exploitation capability of EJaya, what makes this heuristic particularly suitable for problems with a reduced or limited search space.

Benchmark Function	**Formulation**	**Minima**
High-Conditioned Elliptic	$f_1(x) = \sum_{i=1}^{D} (10^6)^{\frac{i-1}{D-1}} x_i^2$	0
Bent-Cigar	$f_2(x) = x_1^2 + 10^6 \sum_{i=2}^{D} x_i^2$	0
Discus	$f_3(x) = 10^6 x_1^2 + \sum_{i=2}^{D} x_i^2$	0
Rosenbrock	$f_4(x) = \sum_{i=1}^{D-1} \left(100\left(x_i^2 - x_{i+1}\right) + (x_i - 1)^2\right)$	0
Ackley's path	$f_5(x) = -20\exp\left(-0.2\sqrt{\frac{1}{D}\sum_{i=1}^{D} x_i^2}\right) - \exp\left(\frac{1}{D}\sum_{i=1}^{D}\cos(2\pi x_i)\right) + 20 + e$	0
Weierstrass	$f_6(x) = \sum_{i=1}^{D}\left(\sum_{k=0}^{k_{max}}\left[a^k \cos\left(2\pi b^k (x_i + 0.05)\right)\right]\right) - D\sum_{k=0}^{k_{max}}\left[a^k \cos\left(\pi b^k\right)\right]$	4
Griewank's Function 8	$f_7(x) = \sum_{i=1}^{D} \frac{x_i^2}{4000} - \prod_{i=1}^{D} \cos\left(\frac{x_i}{\sqrt{i}}\right) + 1$	0
Rastrigin	$f_8(x) = \sum_{i=1}^{D}\left(x_i^2 - 10\cos(2\pi x_i) + 10\right)$	0
Modified Swchefel	$f_9(x) = 418.9829D - \sum_{i=1}^{D} g(z_i),\ z_i = x_i + 4.209687462275036\text{e+}002$	–
	$g(z_i) = \begin{cases} z_i \sin(\sqrt{\lvert z_i\rvert}), & \lvert z_i\rvert \le 500, \\ (500 - \mathrm{mod}(z_i, 500))\sin\left(\sqrt{\lvert 500 - \mathrm{mod}(z_i, 500)\rvert}\right) - \frac{(z_i - 500)^2}{10000D}, & z_i > 500, \\ (\mathrm{mod}(\lvert z_i\rvert, 500) - 500)\sin\left(\sqrt{\lvert \mathrm{mod}(\lvert z_i\rvert, 500) - 500\rvert}\right) - \frac{(z_i + 500)^2}{10000D}, & z_i < -500 \end{cases}$	–
Katsuura	$f_{10}(x) = \frac{10}{D^2}\prod_{i=1}^{D}\left(1 + i\sum_{j=1}^{32}\frac{\lvert 2^j x_i - round(2^j x_i)\rvert}{2^j}\right)^{\frac{10}{D^{1.2}}} - \frac{10}{D^2}$	0
HappyCat	$f_{11}(x) = \left\lvert\sum_{i=1}^{D} x_i^2 - D\right\rvert^{\frac{1}{4}} + \frac{0.5\sum_{i=1}^{D} x_i^2 + \sum_{i=1}^{D} x_i}{D} + 0.5$	–
HGBat	$f_{12}(x) = \left\lvert\left(\sum_{i=1}^{D} x_i^2\right)^2 - \left(\sum_{i=1}^{D} x_i\right)^2\right\rvert^{1/2} + \frac{0.5\sum_{i=1}^{D} x_i^2 + \sum_{i=1}^{D} x_i}{D} + 0.5$	–
Expanded Griewank's plus Rosenbrock's	$f_{13}(x) = f_7(f_4(x_1, x_2)) + f_7(f_4(x_2, x_3)) + \ldots + f_7(f_4(x_{D-1}, x_D)) + f_7(f_4(x_D, x_1))$	
Expanded Schaffer Function 6	$f_{14}(x) = g(x_1, x_2) + g(x_2, x_3) + \ldots + g(x_{D-1}, x_D), g(x_D, x_1)$	–
	$g(x, y) = 0.5 + \frac{\left(\sin^2\left(\sqrt{x^2 + y^2}\right) - 0.5\right)}{\left(1 + 0.001\left(x^2 + y^2\right)\right)^2}$	
Langermann	$f_{15}(x) = -\sum_{i=1}^{5} \frac{c_i \cos\left\{\pi\left[(x_1 - a_i)^2 + (x_2 - b_u)^2\right]\right\}}{e^{\frac{(x_1 - a_i)^2 + (x_2 - b_u)^2}{\pi}}}$	−5.1621259
Eggholder	$f_{16}(x) = -x_1 \sin\left(\sqrt{\lvert x1 - x2 - 47\rvert}\right) - (x_2 + 47)\sin\left(\sqrt{\lvert \frac{1}{2}x_1 + x_2 + 47\rvert}\right)$	−959.64066
Holder's table	$f_{17}(x) = -\left\lvert e^{\left\lvert 1 - \frac{\sqrt{x_1^2 + x_2^2}}{\pi}\right\rvert}\sin(x_1)\cos(x_2)\right\rvert$	–
DropWave	$f_{18}(x) = -\frac{1 + \cos\left(12\sqrt{\sum_i^D x_i^2}\right)}{2 + 0.5\sum_{i=1}^{D} x_i^2}$	0
Bohachevsky	$f_{19}(x) = \sum_i^{D-1}\left[x_i^2 + 2x_{i+1}^2 - 0.3\cos(3\pi x_i) - 0.4\cos(4\pi x_{i+1}) + 0.7\right]$	0
Whitley	$f_{20}(x) = \sum_{i=1}^{D}\sum_{j=1}^{D}\left[\frac{\left(100(x_i^2 - x_j)^2 + (1 - x_j)^2\right)^2}{4\,000} - \cos\left(100\left(x_i^2 - x_j\right)^2 + (1 - x_j)^2\right) + 1\right]$	0

Table 8. Benchmark functions testbed

f	Stats	Jaya	CJaya	EJaya	f	Stats	Jaya	CJaya	EJaya
f01	Mean	2.163e−02	8.794e−06	4.893**e**−11	**f11**	Mean	3.709e−01	1.633e−01	1.456**e**−01
	Best	5.739e−03	1.352e−06	8.380**e**−12		Best	3.011e−01	8.737e−02	5.084**e**−02
	Runtime (s)	2.101e+00	2.081e+00	2.127**e**+00		Runtime (s)	9.867e+00	9.849e+00	1.036**e**+01
	Rank	3	2	**1**		Rank	3	2	**1**
f02	Mean	7.577e+00	2.714e−03	1.559**e**−08	**f12**	Mean	2.411e−01	1.331e−01	1.279**e**−01
	Best	1.907e+00	5.027e−04	1.769**e**−09		Best	1.490e−01	6.492e−02	7.453**e**−02
	Runtime (s)	2.071e+00	2.049e+00	2.151**e**+00		Runtime (s)	9.645e+00	9.525e+00	1.021**e**+01
	Rank	3	2	**1**		Rank	3	2	**1**
f03	Mean	1.436e−04	3.124e−08	1.633**e**−13	**f13**	Mean	1.487e−01	9.338e−02	5.099**e**−02
	Best	2.060e−05	5.067e−09	2.139**e**−14		Best	1.273e−02	4.638e−02	2.637**e**−03
	Runtime (s)	2.081e+00	2.064e+00	2.130**e**+00		Runtime (s)	9.847e+00	9.663e+00	9.967**e**+00
	Rank	3	2	**1**		Rank	3	2	**1**
f04	Mean	8.751**e**+11	9.059e+11	9.203e+11	**f14**	Mean	0.000**e**+00	5.551e−18	0.000e+00
	Best	1.237**e**+11	1.558e+11	2.087e+11		Best	0.000**e**+00	0.000e+00	0.000e+00
	Runtime (s)	2.081**e**+00	2.054e+00	2.128e+00		Runtime (s)	3.057**e**+00	3.085e+00	3.163e+00
	Rank	**1**	2	3		Rank	**1**	3	2
f05	Mean	2.000**e**+01	2.000e+01	2.000e+01	**f15**	Mean	−5.162**e**+00	−5.162**e**+00	−5.162**e**+00
	Best	2.000**e**+01	2.000e+01	2.000e+01		Best	−5.162**e**+00	−5.162**e**+00	−5.162**e**+00
	Runtime (s)	2.084**e**+00	2.092e+00	2.115e+00		Runtime (s)	1.036**e**+01	1.030**e**+01	1.059**e**+01
	Rank	**1**	2	3		Rank	**1**	**2**	**3**
f06	Mean	1.800e+02	1.800e+02	1.800**e**+02	**f16**	Mean	−9.596e+02	−9.596e+02	−9.596**e**+02
	Best	1.800e+02	1.800e+02	1.800**e**+02		Best	−9.596e+02	−9.596e+02	−9.596**e**+02
	Runtime (s)	2.090e+00	2.049e+00	2.123**e**+00		Runtime (s)	2.971e+00	3.094e+00	3.147**e**+00
	Rank	2	3	**1**		Rank	3	2	**1**
f07	Mean	5.802e−01	5.232e−01	4.015**e**−01	**f17**	Mean	−9.140**e**+18	−9.140**e**+18	−9.140**e**+18
	Best	3.096e−01	2.868e−01	2.256**e**−01		Best	−9.140**e**+18	−9.140**e**+18	−9.140**e**+18
	Runtime (s)	2.047e+00	2.059e+00	2.115**e**+00		Runtime (s)	3.068**e**+00	3.040**e**+00	3.066**e**+00
	Rank	3	2	**1**		Rank	**1**	**2**	**3**
f08	Mean	4.250e+01	3.932e+01	2.619**e**+01	**f18**	Mean	−9.362e−01	−9.362e−01	−9.362**e**−01
	Best	2.954e+01	1.975e+01	9.750**e**+00		Best	−9.362e−01	−9.362e−01	−9.362**e**−01
	Runtime (s)	2.084e+00	2.070e+00	2.129**e**+00		Runtime (s)	1.039e+01	1.015e+01	1.058**e**+01
	Rank	3	2	**1**		Rank	2	3	**1**
f09	Mean	1.325e−04	1.273e−04	1.273**e**−04	**f19**	Mean	2.776**e**−17	5.551e−17	3.886e−17
	Best	1.280e−04	1.273e−04	1.273**e**−04		Best	0.000**e**+00	0.000e+00	0.000e+00
	Runtime (s)	2.057e+00	2.068e+00	2.117**e**+00		Runtime (s)	1.038**e**+01	1.016e+01	1.044e+01
	Rank	3	2	**1**		Rank	**1**	3	2
f10	Mean	0.000**e**+00	0.000**e**+00	0.000**e**+00	**f20**	Mean	4.039e+01	4.852e+01	3.604**e**+01
	Best	0.000**e**+00	0.000**e**+00	0.000**e**+00		Best	0.000e+00	0.000e+00	9.894**e**+00
	Runtime (s)	2.095**e**+00	2.110**e**+00	2.149**e**+00		Runtime (s)	1.040e+01	1.031e+01	1.059**e**+01
	Rank	**1**	**2**	**3**		Rank	2	3	**1**

Table 9. Benchmark functions f01-f20 results with Jaya and improvements of Jaya, CJaya and EJaya, $D = 10$

Multimodal Functions Analysis

Multimodal functions have many local minima, and they are more difficult to optimize than unimodal functions. Hence, the end results of this type of functions are more relevant since they mirror the capacity of the strategy to get away from local optima and finding a result close to the global optimum [22]. In this case, although there is no clear supremacy of an algorithm, CJaya and EJaya emerge as the most effective meta-heuristics in optimizing the benchmark functions. This can be observed from Tables 9, 10 and 11 for functions $f_{04} - f_{20}$. CJaya and EJaya are also more scalable than Jaya, since increasing the number of dimensions in the experiments significantly improves their efficiency, and they scale positions in the rank.

f	Stats	Jaya	CJaya	EJaya	f	Stats	Jaya	CJaya	EJaya
f01	Mean	3.005e+02	5.849e+00	1.007**e**−05	**f11**	Mean	1.307e+00	6.960e−01	6.543**e**−01
	Best	2.404e+00	2.258e−03	7.029**e**−07		Best	1.083e+00	4.676e−01	4.874**e**−01
	Runtime (s)	4.572e+01	4.531e+01	4.708**e**+01		Runtime (s)	4.517e+01	4.584e+01	4.736**e**+01
	Rank	3	2	**1**		Rank	3	2	**1**
f02	Mean	3.828e−03	1.674e−05	3.141**e**−09	**f12**	Mean	1.382e+00	6.722**e**−01	6.740e−01
	Best	1.453e−03	3.388e−06	9.776**e**−10		Best	1.125e+00	2.959**e**−01	3.345e−01
	Runtime (s)	4.546e+01	4.536e+01	4.761**e**+01		Runtime (s)	4.567e+01	4.549**e**+01	4.699e+01
	Rank	3	2	**1**		Rank	3	**1**	2
f03	Mean	3.828e−03	1.674e−05	3.141**e**−09	**f13**	Mean	2.396e+04	1.277e+00	3.971**e**−01
	Best	1.453e−03	3.388e−06	9.776**e**−10		Best	1.869e+01	5.963e−01	7.343**e**−03
	Runtime (s)	4.546e+01	4.536e+01	4.761**e**+01		Runtime (s)	4.551e+01	4.522e+01	4.740**e**+01
	Rank	3	2	**1**		Rank	3	2	**1**
f04	Mean	4.979**e**+12	4.988e+12	4.987e+12	**f14**	Mean	2.776e−18	0.000**e**+00	5.551e−18
	Best	4.747**e**+12	4.749e+12	4.708e+12		Best	0.000e+00	0.000**e**+00	0.000e+00
	Runtime (s)	4.568**e**+01	4.514e+01	4.686e+01		Runtime (s)	3.013e+00	3.016**e**+00	3.057e+00
	Rank	**1**	3	2		Rank	2	**1**	3
f05	Mean	2.000e+01	2.000**e**+01	2.000e+01	**f15**	Mean	−5.162**e**+00	−5.162**e**+00	−5.162**e**+00
	Best	2.000e+01	2.000**e**+01	2.000e+01		Best	−5.162**e**+00	−5.162**e**+00	−5.162**e**+00
	Runtime (s)	4.612e+01	4.568**e**+01	4.732e+01		Runtime (s)	4.538**e**+01	4.533**e**+01	4.698**e**+01
	Rank	3	**1**	2		Rank	**1**	**2**	**3**
f06	Mean	4.900e+03	4.900**e**+03	4.900e+03	**f16**	Mean	−9.589e+02	−9.580e+02	−9.596**e**+02
	Best	4.900e+03	4.900**e**+03	4.900e+03		Best	−9.596e+02	−9.596e+02	−9.596**e**+02
	Runtime (s)	4.540e+01	4.494**e**+01	4.693e+01		Runtime (s)	2.936e+00	2.976e+00	3.026**e**+00
	Rank	3	**1**	2		Rank	2	3	**1**
f07	Mean	2.628e−02	1.441e−02	4.800**e**−03	**f17**	Mean	−9.140**e**+18	−9.140**e**+18	−9.140**e**+18
	Best	2.090e−05	6.990e−08	3.203**e**−11		Best	−9.140**e**+18	−9.140**e**+18	−9.140**e**+18
	Runtime (s)	4.655e+01	4.605e+01	4.724**e**+01		Runtime (s)	2.972**e**+00	2.926**e**+00	3.075**e**+00
	Rank	3	2	**1**		Rank	**1**	**2**	**3**
f08	Mean	4.780e+02	4.330e+02	4.191**e**+02	**f18**	Mean	−5.885e−03	−2.969e−02	−1.862**e**−01
	Best	3.802e+02	3.059e+02	3.465**e**+02		Best	−1.651e−02	−7.774e−02	−2.888**e**−01
	Runtime (s)	4.689e+01	4.648e+01	4.794**e**+01		Runtime (s)	4.544e+01	4.550e+01	4.711**e**+01
	Rank	3	2	**1**		Rank	3	2	**1**
f09	Mean	2.768e+01	9.228e+00	6.364**e**−04	**f19**	Mean	7.752e+00	5.088e+00	2.512**e**+00
	Best	7.062e−04	6.367e−04	6.364**e**−04		Best	2.554e+00	4.720e−01	3.801**e**−07
	Runtime (s)	4.636e+01	4.554e+01	4.737**e**+01		Runtime (s)	4.607e+01	4.565e+01	4.716**e**+01
	Rank	3	2	**1**		Rank	3	2	**1**
f10	Mean	4.231e−05	0.000**e**+00	0.000**e**+00	**f20**	Mean	1.998e+05	2.250e+03	2.184**e**+03
	Best	0.000e+00	0.000**e**+00	0.000**e**+00		Best	1.715e+03	1.256e+03	1.940**e**+03
	Runtime (s)	4.550e+01	4.529**e**+01	4.705**e**+01		Runtime (s)	4.536e+01	4.550e+01	4.684**e**+01
	Rank	3	**1**	**2**		Rank	3	2	**1**

Table 10. Benchmark functions f01-f20 results with Jaya and improvements of Jaya, CJaya and EJaya, $D = 50$

From this analysis, it is noteworthy that CJaya and EJaya obtain better or equal (when all the algorithms reach the global minimum) results than Jaya more than half of the times: for $D = 10$ this condition occurs in 14 functions, and for $D = 50$ and $D = 100$ this occurs in 17 functions out of 20. Moreover, EJaya reaches the top position in the rank in most cases, sometimes with a lead of several orders of magnitude over Jaya, and most of the time providing an improvement between 20 and 60 percentage points over. Although statistical tests results are to be presented in the next section, it can be concluded in view of these experiments that both CJaya as EJaya, and especially the latter, offer an improvement in the performance of Jaya, with very small increases in runtime (around 5 % in the case of EJaya).

f	Stats	Jaya	CJaya	EJaya	f	Stats	Jaya	CJaya	EJaya
f01	Mean	9.525e+03	1.689e+00	1.147**e**−04	**f11**	Mean	1.581e+00	7.363**e**−01	7.407e−01
	Best	1.978e+01	3.283e−02	4.053**e**−06		Best	1.377e+00	5.524**e**−01	6.278e−01
	Runtime (s)	1.792e+02	1.775e+02	1.866**e**+02		Runtime (s)	1.796e+02	1.789**e**+02	1.872e+02
	Rank	3	2	**1**		Rank	3	**1**	2
f02	Mean	1.621e+04	7.249e+00	7.974**e**−03	**f12**	Mean	1.574e+00	7.183e−01	7.068**e**−01
	Best	2.506e+03	1.578e+00	1.622**e**−03		Best	1.405e+00	3.325e−01	3.416**e**−01
	Runtime (s)	1.788e+02	1.784e+02	1.872**e**+02		Runtime (s)	1.795e+02	1.834e+02	1.938**e**+02
	Rank	3	2	**1**		Rank	3	2	**1**
f03	Mean	3.689e−02	2.463e−05	1.684**e**−08	**f13**	Mean	1.437e+07	1.281e+01	1.682**e**+00
	Best	6.813e−03	4.110e−06	5.686**e**−09		Best	3.025e+05	4.059e+00	1.359**e**+00
	Runtime (s)	1.793e+02	1.776e+02	1.870**e**+02		Runtime (s)	1.802e+02	1.776e+02	1.848**e**+02
	Rank	3	2	**1**		Rank	3	2	**1**
f04	Mean	1.000e+13	9.936**e**+12	1.003e+13	**f14**	Mean	2.776e−18	0.000**e**+00	2.776e−18
	Best	9.701e+12	9.527**e**+12	9.524e+12		Best	0.000e+00	0.000**e**+00	0.000e+00
	Runtime (s)	1.811e+02	1.803**e**+02	1.845e+02		Runtime (s)	5.908e+00	5.886**e**+00	6.021e+00
	Rank	2	**1**	3		Rank	2	**1**	3
f05	Mean	2.001e+01	2.000**e**+01	2.000e+01	**f15**	Mean	−5.162**e**+00	−5.162**e**+00	−5.162**e**+00
	Best	2.000e+01	2.000**e**+01	2.000e+01		Best	−5.162**e**+00	−5.162**e**+00	−5.162**e**+00
	Runtime (s)	1.803e+02	1.798**e**+02	1.852e+02		Runtime (s)	1.802**e**+02	1.798**e**+02	1.847**e**+02
	Rank	3	**1**	2		Rank	**1**	**2**	**3**
f06	Mean	1.980e+04	1.980e+04	1.980**e**+04	**f16**	Mean	−9.558e+02	−9.580e+02	−9.596**e**+02
	Best	1.980e+04	1.980e+04	1.980**e**+04		Best	−9.596e+02	−9.596e+02	−9.596**e**+02
	Runtime (s)	1.800e+02	1.780e+02	1.846**e**+02		Runtime (s)	6.016e+00	5.971e+00	6.005**e**+00
	Rank	3	2	**1**		Rank	3	2	**1**
f07	Mean	7.049e−03	3.078e−03	2.772**e**−03	**f17**	Mean	−9.140**e**+18	−9.140**e**+18	−9.140**e**+18
	Best	6.537e−05	2.726e−08	3.404**e**−11		Best	−9.140**e**+18	−9.140**e**+18	−9.140**e**+18
	Runtime (s)	1.802e+02	1.784e+02	1.852**e**+02		Runtime (s)	5.861**e**+00	5.918**e**+00	6.016**e**+00
	Rank	3	2	**1**		Rank	**1**	**2**	**3**
f08	Mean	1.100e+03	7.472**e**+02	9.940e+02	**f18**	Mean	−5.684e−04	−2.680e−03	−1.271**e**−02
	Best	6.421e+02	4.371**e**+02	4.835e+02		Best	−9.462e−04	−5.352e−03	−3.199**e**−02
	Runtime (s)	1.789e+02	1.785**e**+02	1.858e+02		Runtime (s)	1.780e+02	1.770e+02	1.841**e**+02
	Rank	3	**1**	2		Rank	3	2	**1**
f09	Mean	3.285**e**−03	1.846e+01	3.691e+01	**f19**	Mean	2.986e+01	2.553e+01	1.492**e**+01
	Best	1.809**e**−03	1.273e−03	1.273e−03		Best	1.966e+01	1.680e+01	8.398**e**+00
	Runtime (s)	1.790**e**+02	1.776e+02	1.840e+02		Runtime (s)	1.782e+02	1.781e+02	1.857**e**+02
	Rank	**1**	2	3		Rank	3	2	**1**
f10	Mean	2.367e−04	1.637**e**−07	2.447e−06	**f20**	Mean	3.600e+08	9.062e+03	7.475**e**+03
	Best	0.000e+00	0.000**e**+00	0.000e+00		Best	8.277e+06	4.638e+03	9.412**e**+02
	Runtime (s)	1.806e+02	1.783**e**+02	1.852e+02		Runtime (s)	1.784e+02	1.777e+02	1.843**e**+02
	Rank	3	**1**	2		Rank	3	2	**1**

Table 11. Benchmark functions f01-f20 results with Jaya and improvements of Jaya, CJaya and EJaya, $D = 100$

Statistical Tests

The best known procedure for multiple comparison to check differences between more than two related samples is the Friedman Ranks test [5]. Given a certain value of statistical significance limit (generally, p-value = 0.05) it is determined whether the null hypothesis H0 (H0: algorithms have a similar behaviour) can be rejected. On the other hand, it must be born in mind that the main drawback is that Friedman Ranks test can only detect significant differences with respect to any multiple comparison, but it is unable to establish appropriate comparisons between some considered algorithms. When the goal is not only to know whether there are differences between the methods, but to compare which relations have more statistical significance, a series of ad-hoc hypotheses using a post-hoc test must be done after the Friedman Ranks test. In this work, the Friedman Ranks Post-Hoc test

is used to get the unadjusted p-values that allow the comparison of the statistical significance among all the implemented algorithms. The Friedman Ranks Post-Hoc test obtains a set of p-values that determine the degree of rejection of each null hypothesis for each pair of algorithms. It must be noted that for this analysis it is not possible to make a comparison between pairs by type, such as in the Wilcoxon test because they involve a set of results where the Family-Wise Error Rate (FWER) is not controlled.

The Friedman Ranks and Friedman Ranks Post-Hoc tests are conducted in this work for the set of algorithms in the case of 10, 50 and 100 dimensions for average value, better results as well as runtime.

Stats	**D = 10**	**D = 50**	**D = 100**
Mean	**3.379e−04**	**1.008e−04**	**4.800e−04**
Best	**2.410e−02**	**5.615e−04**	**2.390e−02**
Runtime	**6.772e−07**	**7.126e−08**	**1.219e−08**

Table 12. Friedman Ranks test p-values for mean, best and runtime results

First, in Table 12, Friedman Ranks test results are presented. It can be observed that for all the cases the Friedman Ranks test indicates that the null hypothesis should be rejected, that is, there are statistically significant differences (considering a statistical significance limit of p-value = 0.05) between the strategies for the different analysed results (i.e., mean, best and runtime) and dimensions (i.e., 10, 50 and 100), so it makes sense to conduct a Friedman Ranks Post-Hoc test in each case. Table 13 shows p-values for 10, 50 and 100 dimensions, for all the parameters to be analysed and for all the possible comparisons. First, regarding mean results, it is observed that EJaya improves Jaya with statistical significance in all dimensions while CJaya improves Jaya with statistically significance for medium and high dimensions, 50 and 100, respectively. In addition, it can be seen that EJaya and CJaya show similar results in a statistical significance sense. As for the best results analysis, the statistical significance corroborates the results related to the mean, being EJaya once again the strategy presenting the best behaviour. Finally, as expected, the introduction of modifications to achieve coherent attraction and repulsion factors and restrained intensity for flight slows down Jaya variants as more control checks are introduced. In this way, it can be observed that statistically significant differences are found in runtime between EJaya and both Jaya and CJaya, except for high dimensions case (i.e., 100), no statistical significance is found between CJaya and Jaya.

6 CONCLUSIONS

In recent years, several meta-heuristics have been proposed and research in this sense is still in the spotlight due to the wide range of applications requiring more efficient optimization procedures in industrial and scientific areas. A major issue in the implementation and comparison of meta-heuristics is given by their simplicity, which

Stats	Comparison	D = 10	D = 50	D = 100
Mean	EJaya-CJaya	2.305e−01	1.709e−01	8.521e−01
	Jaya-CJaya	6.895e−02	**5.116e−02**	**3.522e−03**
	Jaya-EJaya	**4.097e−04**	**9.807e−05**	**5.1319e−04**
Best	EJaya-CJaya	5.882e−01	5.891e−01	4.602e−01
	Jaya-CJaya	2.302e−01	1.656**e**−02	3.097**e**−03
	Jaya-EJaya	**2.391e−02**	**5.763e−04**	**3.070e−05**
Runtime	EJaya-CJaya	**5.571e−07**	**7.333e−08**	**1.051e−08**
	Jaya-CJaya	3.289e−01	2.537e−01	**1.965e−02**
	Jaya-EJaya	**4.184e−04**	**2.468e−04**	**4.474e−03**

Table 13. Friedman Ranks Post-Hoc test p-values for mean, best and runtime results

is directly related to the number of parameters and their associated interactions. Jaya algorithm is a recent and simple meta-heuristic offering better or comparable results to a large set of major meta-heuristics nowadays such as GA, PSO, DE, ABC and TLBO. In this work, Jaya algorithm is improved to offer a more efficient specific parameter-free meta-heuristic of one phase, EJaya. EJaya includes coherent attraction and repulsion movements and restrained intensity for flights that overcome limitations of Jaya and the contributions of these improvements to general success are analysed gradually (CJaya and EJaya) to valid each of the suggested incremental solutions. The proposal has been tested on a set of various standard benchmark functions from CEC. The results obtained by EJaya offer more efficient results than Jaya in terms of mean and best accomplishments, as proved by statistical tests showing statistical significance, a higher exploitation capability and scalability. Finally, it must be highlighted that EJaya does not require tuning of algorithm specific parameters what may be beneficial to applications where the setup of optimization may be critical for the whole performance of the systems. Hence, this work presents one more step towards the development of simpler and fast optimization strategies.

In future works, EJaya will be applied to a problem of practical importance nowadays that can greatly benefit from the simplicity and speed of EJaya in its set-up. Specifically, EJaya will be considered for the learning of Fuzzy Rule-Based Schedulers in Cloud Computing. Cloud Computing is a very dynamic environment where a fast knowledge acquisition strategy can achieve significant improvement in terms of time and power saving in the allocation of workload among the large number of involved computational resources. Further, results will be compared to other meta-heuristics.

Acknowledgments

This work was financially supported by the Research Projects TEC2015-67387-C4-2 and 2011-TIC-7278.

REFERENCES

[1] Bindiya, T. S.—Elias, E.: Meta-Heuristic Evolutionary Algorithms for the Design of Optimal Multiplier-Less Recombination Filter Banks. Information Sciences, Vol. 339, 2016, pp. 31–52, doi: 10.1016/j.ins.2015.12.018.

[2] Chinta, S.—Kommadath, R.—Kotecha, P.: A Note on Multi-Objective Improved Teaching-Learning Based Optimization Algorithm (MO-ITLBO). Information Sciences, Vol. 373, 2016, pp. 337–350, doi: 10.1016/j.ins.2016.08.061.

[3] Chou, J.-S.—Ngo, N.-T.: Time Series Analytics Using Sliding Window Metaheuristic Optimization-Based Machine Learning System for Identifying Building Energy Consumption Patterns. Applied Energy, Vol. 177, 2016, pp. 751–770, doi: 10.1016/j.apenergy.2016.05.074.

[4] Deb, K.—Agrawal, S.: Understanding Interactions Among Genetic Algorithm Parameters. Foundations of Genetic Algorithms V, Morgan Kaufmann, 1999, pp. 265–286.

[5] Derrac, J.—García, S.—Molina, D.—Herrera, F.: A Practical Tutorial on the Use of Nonparametric Statistical Tests as a Methodology for Comparing Evolutionary and Swarm Intelligence Algorithms. Swarm and Evolutionary Computation, Vol. 1, 2011, No. 1, pp. 3–18, doi: 10.1016/j.swevo.2011.02.002.

[6] García-Galán, S.—Prado, R. P.—Expósito, J. E. M.: Swarm Fuzzy Systems: Knowledge Acquisition in Fuzzy Systems and Its Applications in Grid Computing. IEEE Transactions on Knowledge and Data Engineering, Vol. 26, 2014, No. 7, pp. 1791–1804, doi: 10.1109/TKDE.2013.118.

[7] García-Galán, S.—Prado, R. P.—Expósito, J. E. M.: Rules Discovery in Fuzzy Classifier Systems with PSO for Scheduling in Grid Computational Infrastructures. Applied Soft Computing, Vol. 29, 2015, pp. 424–435, doi: 10.1016/j.asoc.2014.11.064.

[8] Gendreau, M.—Potvin, J.-Y.: Handbook of Meta-Heuristics. Springer US, 2010.

[9] Goldberg, D. E.: Genetic Algorithms in Search, Optimization and Machine Learning. 1st Edition. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1989.

[10] Jordehi, A. R.: Enhanced Leader PSO (ELPSO): A New PSO Variant for Solving Global Optimisation Problems. Applied Soft Computing, Vol. 26, 2015, pp. 401–417, doi: 10.1016/j.asoc.2014.10.026.

[11] Karaboga, D.—Basturk, B.: A Powerful and Efficient Algorithm for Numerical Function Optimization: Artificial Bee Colony (ABC) Algorithm. Journal of Global Optimization, Vol. 39, 2007, No. 3, pp. 459–471, doi: 10.1007/s10898-007-9149-x.

[12] Karaboga, D.—Basturk, B.: On the Performance of Artificial Bee Colony (ABC) Algorithm. Applied Soft Computing, Vol. 8, 2008, No. 1, pp. 687–697, doi: 10.1016/j.asoc.2007.05.007.

[13] Kennedy, J.—Eberhart, R.: Particle Swarm Optimization. Proceedings of the IEEE International Conference on Neural Networks (ICNN'95), Vol. 4, 1995, pp. 1942-1948, doi: 10.1109/ICNN.1995.488968.

[14] Kennedy, J.—Eberhart, R. C.—Shi, Y.: Swarm Intelligence. Springer, 2001.

[15] Liang, J. J.—Qu, B. Y.—Suganthan, P.: Problem Definitions and Evaluation Criteria for the CEC 2014 Special Session and Competition on Single Objective Real-Parameter Numerical Optimization. Technical Report, Zhengzhou University, China and Nanyang Technological University, Singapore, 2013, IIT Kanpur, India, 2014.

[16] Liu, S. Q.—Kozan, E.: A Hybrid Metaheuristic Algorithm to Optimise a Real-World Robotic Cell. Computers and Operations Research, Vol. 84, 2017, pp. 188–194, doi: 10.1016/j.cor.2016.09.011.

[17] Mesejo, P.—Ibáñez, Ó.—Cordón, Ó.—Cagnoni, S.: A Survey on Image Segmentation Using Metaheuristic-Based Deformable Models: State of the Art and Critical Analysis. Applied Soft Computing, Vol. 44, 2016, pp. 1–29, doi: 10.1016/j.asoc.2016.03.004.

[18] Minetti, G.—Leguizamón, G.—Alba, E.: An Improved Trajectory-Based Hybrid Metaheuristic Applied to the Noisy DNA Fragment Assembly Problem. Information Sciences, Vol. 277, 2014, pp. 273–283, doi: 10.1016/j.ins.2014.02.020.

[19] Patel, V. K.—Savsani, V. J.: Heat Transfer Search (HTS): A Novel Optimization Algorithm. Information Sciences, Vol. 324, 2015, pp. 217–246, doi: 10.1016/j.ins.2015.06.044.

[20] Rao, R. V.: Jaya: A Simple and New Optimization Algorithm for Solving Constrained and Unconstrained Optimization Problems. International Journal of Industrial Engineering Computations, Vol. 7, 2016, No. 1, pp. 19–34, doi: 10.5267/j.ijiec.2015.8.004.

[21] Rao, R. V.: Teaching-Learning-Based Optimization Algorithm. Springer International Publishing, Cham, 2016, pp. 9–39.

[22] Rashedi, E.—Nezamabadi-Pour, H.—Saryazdi, S.: GSA: A Gravitational Search Algorithm. Information Sciences, Vol. 179, 2009, No. 13, pp. 2232–2248, doi: 10.1016/j.ins.2009.03.004.

[23] Storn, R.—Price, K.: Differential Evolution – A Simple and Efficient Heuristic for Global Optimization over Continuous Spaces. Journal of Global Optimization, Vol. 11, 1997, No. 4, pp. 341–359, doi: 10.1023/A:1008202821328.

[24] Wolpert, D. H.—Macready, W. G.: No Free Lunch Theorems for Optimization. IEEE Transactions on Evolutionary Computation, Vol. 1, 1997, No. 1, pp. 67–82.

[25] Xiong, Y.—Golden, B.—Wasil, E.: A One-Parameter Genetic Algorithm for the Minimum Labeling Spanning Tree Problem. IEEE Transactions on Evolutionary Computation, Vol. 9, 2005, No. 1, pp. 55–60, doi: 10.1109/4235.585893.

[26] Xu, J.—Kelly, J. P.: A Network Flow-Based Tabu Search Heuristic for the Vehicle Routing Problem. Transportation Science, Vol. 30, 1996, No. 4, pp. 379–393, doi: 10.1287/trsc.30.4.379.

[27] Zhang, L.—Wong, T. N.: Solving Integrated Process Planning and Scheduling Problem with Constructive Metaheuristics. Information Sciences, Vol. 340-341, 2016, pp. 1–16, doi: 10.1016/j.ins.2016.01.001.

[28] Zhu, Q.—Lin, Q.—Du, Z.—Liang, Z.—Wang, W.—Zhu, Z.—Chen, J.—Huang, P.—Ming, Z.: A Novel Adaptive Hybrid Crossover Operator for Multiobjective Evolutionary Algorithm. Information Sciences, Vol. 345, 2016, pp. 177–198, doi: 10.1016/j.ins.2016.01.046.

Rafael Rodríguez-Reche received his B.Eng. degree in computer science engineering from University of Jaén, Jaén, Spain, in 2014 and his M.Sc. degree in telecommunication engineering in 2016. At present, he collaborates as a researcher at the Telecommunication Engineering Department of the University of Jaén. His current research interests include blockchain, artificial intelligence, cloud computing, scheduling, machine learning and optimization.

Rocío P. Prado received her M.Sc. degree in telecommunication engineering from Seville University, Seville, Spain, in 2008 and her Ph.D. degree in telecommunication engineering with European Mention from University of Jaén, Jaén, Spain, in 2011. At present, she is Associate Professor with the Telecommunication Engineering Department of the Jaén University. Her current research interests include artificial intelligence, machine learning, grid/cloud computing and scheduling. She is an active member of the research group "Signal Processing for Telecommunication Systems" (TIC-188 of the PAI) and she is in the editorial board of 19 JCR-indexed international journals such as IEEE Transactions on Fuzzy Systems, Applied Soft Computing and IEEE Transactions on Data and Knowledge Engineering.

Sebastián García-Galán received his M.Sc. and Ph.D. degrees in telecommunication engineering from the Málaga University and the Technical University of Madrid (UPM), in 1995 and 2004, respectively. Since 1999, he is Associate Professor at the Telecommunication Engineering Department of the Jaén University. He is a member of the research group "Signal Processing for Telecommunication Systems" (TIC-188 of the PAI) and the European Society for Fuzzy Logic And Technology (EUSFLAT). His areas of research interest are artificial intelligence, grid/cloud computing, speech and audio analysis. Also, he is a reviewer of several journals indexed in JCR. He is involved in research projects of the Spanish Ministry of Science and Education and of private companies.

José Enrique Muñoz-Expósito received his M.Sc. degree in telecommunication engineering from Málaga University, Spain and Ph.D. in telecommunication engineering from Jaén University, Spain. Since 2003, he has been Associate Professor at the Telecommunication Engineering Department of the Jaén University. His research interests include speech and audio analysis, artificial intelligence, grid and cloud computing. He is currently Senior Researcher with the chair for "Signal Processing and Telecommunication Systems" (TIC-188 of the PAI). He is involved in research projects and works also for the editorial reviewer board of the Annual Telematics Engineering Conferences (JITEL).

Nicolás Ruiz-Reyes received his M.Sc. and Ph.D. degrees in telecommunication engineering from the Technology University of Madrid and the University of Alcalá, in 1993 and 2001, respectively. Since 2010 he has been Full Professor at the Telecommunication Engineering Department of the University of Jaén, he is also the head of the research group TIC-188-PAI. He is a member of the IEEE Signal Processing Society and Audio Engineering Society. His areas of research interest are signal processing and its applications to communications. He is involved in research projects of the Spanish Ministry of Science, European Commission and private companies.

Computing and Informatics, Vol. 38, 2019, 291–320, doi: 10.31577/cai_2019_2_291

AN EXPERIMENTAL STUDY ON VIRTUAL MACHINE LIVE MIGRATION IMPACT ON SERVICES PERFORMANCE

Petrônio Bezerra, Marcela Santos, Edlane Alves
Anderson Costa

Federal Institute of Paraíba (IFPB)
Campina Grande Campus
Tranquilino Coelho Lemos, 671 – Dinamérica
58432-300 Campina Grande, PB – Brazil
e-mail: {petroniocg, anderson}@ifpb.edu.br

Fellype Albuquerque, Gustavo Martins, Reinaldo Gomes

Federal University of Campina Grande (UFCG)
Department of Systems and Computation
Aprígio Veloso, 882 – Universitário
58429-900 Campina Grande, PB – Brazil
e-mail: gustavo.martins@copin.ufcg.edu.br, reinaldo@dsc.ufcg.edu.br

Abstract. One important benefit of servers' virtualization is the reduction of the maintenance complexity of infrastructures. A key feature is servers' live migration which allows virtual servers to be exchanged between physical machines without stopping their services. However, virtualization also has some drawbacks caused by the overhead generated. Our research evaluated live migration process overhead, on real and virtual environments, noticed from the client's side regarding two different services: web and database. YCSB and *ab* Benchmark were adopted as workloads. Almost all tests on real environment overcame those on virtual, with both benchmarks. The impact of the live migration in the services was evident, proving to be more effective on real machines than on virtual machines. We found the DB service accommodated better to the virtual environment and to migration than Web service. We also considered an environment with multiple migrations which presented a higher degradation than when only one migration is performed.

Keywords: Migration, virtualization, overhead, evaluation, Xen

Mathematics Subject Classification 2010: 62-J02

1 INTRODUCTION

Cloud computing is currently the most important technology to support on-demand allocation and delivery of computing resources. These resources can be servers, storage systems, networking, databases, etc. These cloud computing resources are organized in levels as a service aiming to better address customers' requirements. In the market view, cloud computing in recent years has had a rapid growth, as highlighted in [1] presenting the US spending projection in cloud computing between 2010–2015. An annual growth of around 40 % was predicted, reaching an investment of USD 7 billion. Around that period, analysts predicted a growth in cloud's global market estimated at USD 95 billion in addition to 12 % of software migration to the cloud market. At this point, these growths have driven the research community to investigate topics concerning cloud computing.

According to [2], it is possible to migrate a computational infrastructure to a remote location with a minimal impact on system performance. This is one of the advantages obtained with the use of cloud computing. Many companies have migrated their computing infrastructures to cloud environments because of the benefits of virtualization [3]. However, it is necessary to know the performance of services deployed in Virtual Machines (VMs) when they are running in a cloud computing environment [5]. When you migrate an IT infrastructure to a public cloud, knowledge of how your services can be impacted is crucial. It is worth noting that in cloud environments the goal is often to save energy and make the best use of available resources [4]. These goals may be antagonistic in relation to the better performance of the services deployed in VM.

A technology commonly adopted by cloud computing providers is virtualization [2, 8]. This approach involves a software layer, located between the hardware and Operating System (OS), which is called a Virtual Machine Monitor (VMM) or hypervisor [6]. It allows running different VMs with different OS on a single physical machine at the same time. There are many advantages obtained through virtualization, among them we can quote:

1. Server consolidation: The possibility of running many virtualized servers at the same time in a single physical server facilitates financial savings in acquisition and hardware maintenance.
2. Energy savings: Each instance of VM running in a server represents a physical machine, thus, as many more VMs can be running on single server, the power consumption would be less compared to installing new physical machines to run new services.

3. Load balance: The number of VMs on a single physical server should not compromise the level of services offered by virtualized servers.

4. Easy management: VMs adoptions facilitate server management, for instance, in backup procedures.

5. Migration: A feature reached by virtualization, migration allows movement of a server between different hosts [4].

This procedure can be undertaken without interference or downtime to its services, which is called live migration.

Live migration itself brings many benefits to a cloud provider's environment. For example, the reallocation VM procedure, which uses server's migration, allows better usage of a physical machine, savings in resources and maximizing profit. Nevertheless, despite the benefits of virtualization, overhead generated through virtualization layer has already been the target of many investigations. Regardless of migration, such studies show the impact on the performance of the services hosted on virtualized servers [7].

In this paper, we are interested in checking the impact perceived by a client host accessing services, such as web server and DBMS, running on a virtual server during a live virtual machine migration process, in contrast to most investigations about virtualization overhead which only adopted benchmarks that run on the server side, consequently only checking the overhead effect on server side. The goal of our research is to analyze the overhead effect on the client's side, contrary to what has been proposed by other investigations. This is important because the users through their hosts (clients) are the most interested and affected by services performance. Furthermore, these overhead effects were observed during a live migration procedure. We believe that our findings can help decision makers in cloud computing better address their requirements and to decide if/when a migration might be performed to reduce its impact.

The remainder of the article is organized as follows: Section 2 presents techniques responsible for performing the virtualization process. In Section 3 we have the related works. Then in Section 4 we describe the experiment setup and materials. The analysis of the data is in Section 5. We move on with performance evaluation based on our experiment setup in Section 6. In Section 7, we conclude this paper and discuss the future work.

2 RESOURCE VIRTUALIZATION

There are some techniques responsible for performing the virtualization process according to the way how physical resources will be accessed and allocated to the virtual machines.

2.1 Virtualization Techniques

These techniques allow the isolation and abstraction of the underlying hardware and lower level functions [11]. Different approaches are:

Full Virtualization: Here, the VMM runs on top of a host operating system similarly to other applications, in the user space. In this case, the entire physical platform (CPU, memory, disk, etc.) is being virtualized. The overhead caused by this form of virtualization can be quite significant.

OS-Layer Virtualization: In this approach, the guests' OS is being virtualized instead of the hardware. The virtualizations here occur by running more instances of the same OS in parallel. Because of that, the VMs must use the same kernel as the host OS, not being possible to virtualize a different OS. This is a restriction of this approach.

Paravirtualization: The main difference from the full virtualization is that, in paravirtualization, the guest's OS must be modified. This technique allows specific guest machines to communicate directly with the hardware, rather than communicating with the VMM. For this reason, it offers better performance but has the restriction of having to modify the guest OS.

2.2 Strategies for VM Migration

The VM migration is a very important characteristic for cloud computing environments. The two main ways to perform VM migration are:

Stop-and-copy (or non-live migration): In this approach, the VM is suspended on the source host, it migrates to the destination host through copies of the memory pages and other necessary information, and then the VM is activated again at its destination. It is a more simple way to perform the migration procedure, besides being faster than live migration. However, it presents greater downtime of the applications running on the virtualized server.

Live migration: In live migration, there are several iterations of copying the memory pages from the source host to the destination host. During this process, the services deployed in the VM are still in operation. In this way, the downtime of the applications is minimized and being more interesting in some situations. However, the total migration time is greater in this approach than in stop-and-copy.

3 RELATED WORK

The virtualization theme has been very exploited in cloud computing community, and the performance analysis is a major subject of these investigations. The most employed strategy to compare hypervisors is to apply series of benchmark software to

test a wide range of system devices, for example I/O, disk, memory, networking and processors [10, 12, 13, 14]. Although being the excellent sources of research, none of the cited studies took into consideration the performance evaluation of a specific application running on VM. In those articles, the authors use benchmark software to test a device of the evaluated system.

The following two studies considered the performance evaluation of a specific application running on VM.

In [9], they evaluated two virtualization technologies, Xen and OpenVZ, comparing both technologies to a base system in terms of application performance, resource consumption and scalability, among others. The workload used in this study was RUBiS. They found that the average response time could increase over 400 % in Xen and 100 % in OpenVZ as the number of application instances grew from one to four.

Another similar study using databases running on VM, and focused on the impact of virtualization layer, can be found in [15]. In this study, the overhead of the virtualization layer was evaluated in different databases deployed in VMs, Cassandra and MongoDB, the overhead was observed to consider different virtualization techniques, full virtualization and paravirtualization, when compared to a physical host. The authors executed the YCSB benchmark to evaluate databases' performances. In the findings, virtualization technique was the factor that showed a higher influence over databases' performance compared to that obtained from a physical host. MongoDB reached better performance in most of scenarios.

None of those articles developed a performance analysis of applications running on VMs during the occurrence of a VM live migration process. The other studies in sequence did that.

Another interesting paper, similar to what has been developed, can be found in [16]. In this study, the VM migration was evaluated regarding performance during migration, performance of cloud architecture during VM migration and the energy cost of real-time migration. The goal was to understand how the cloud architecture would respond and deal with real time migrations. The results obtained showed how a virtual machine performs during a live migration. This differs from our proposal in two aspects: we are not interested in evaluating the energy cost and our study was made without the use of the cloud.

In [17], the authors demonstrated how resource consumption and latency can be substantially reduced, allowing better VMs migration performance. Initially, they experimentally studied the factors that contributed to the growth of these two variables on migration. They proposed an alternative technique of remote access memory which significantly reduces the overhead on the migration of VMs. Through simulations and experiments, the authors reduced the overhead in the migration of VMs, resulting in improvements in energy and resource efficiency over the techniques that already exist.

The main distinction of our work is that our objective is to evaluate the impact, from the client's point of view, that a live migration process cause in applications running on VM.

4 EXPERIMENT SETUP

This paper investigates the overhead caused by a live migration of a server in two common applications on the client's side. Like many other studies, we also use benchmarks tools to measure performance. Most research that uses benchmarks do this on server's side, instead, here benchmark software it was used on client's side. This is a differential from other researches.

In this experiment, aspects of performing a VM live migration were evaluated. The aspect concerned was the performance of the services, perceived by a client when a live migration is running on a server. To support machines virtualization and migration Xen hypervisor[1] was used. Xen is an open source hypervisor which enables to use full virtualization and paravirtualization techniques. In our study, only the paravirtualization technique was used, in which guest systems know they are being virtualized because their kernels need to be modified, improving the performance achieved compared to full virtualization approach [9]. Xen was chosen as a virtualization platform for our experiment because it is an open source platform, commonly adopted in investigations concerning live migration [7, 9, 10].

The services used were Apache Web Service™[2] and the Apache Cassandra™[3] a Database Management System (DBMS). Two benchmarks (*ab* and YCSB) were used in our experiments to fulfill a series of performance tests, they are described below:

Apache HTTP Server Benchmarking Tool (*ab*)[4]: Is a tool for benchmarking Apache Hypertext Transfer Protocol (HTTP) servers. It is designed to test how the Apache installation performs. In our experiments, it was used with the objective of generating workload to the VMs on Xen servers. It is important to highlight that this benchmark will simulate the requests made by a user to a web server. With *ab*, it was measured the mean number of requests per second.

Yahoo! Cloud Serving Benchmark (YCSB)[5]: This benchmark aims to generate workload for non-relational database (NoSQL). It is possible to evaluate the performance of the database through information such as average latency and throughput. In our experiments the Apache Cassandra NoSQL database was used. With YCSB the number of operations per second was measured using the Workload option, which makes 50 % of load operations and 50 % of select and update operations.

Initially, the experiments were divided into phases – with environment totally virtualized, with only real machines and with multiple migrations.

[1] `http://www.xenproject.org/`
[2] `http://www.apache.org/`
[3] `http://cassandra.apache.org/`
[4] `https://httpd.apache.org/docs/2.4/programs/ab.html`
[5] `https://github.com/brianfrankcooper/YCSB/wiki`

A. Virtualized Servers

1. *The Environment:* It consisted of one real machine which contained all the virtualized hosts. Figure 1 gives a representation of the simulation environment. It is possible to see the host machine where whole experiments of this phase took place. This host contained four VMs: XenServer1, XenServer2, Client, Network File System (NFS) Server and a Virtual Ethernet Switch. These VMs were virtualized by the VMware Workstation Player[6] free hypervisor.

   The concept of nested virtualization is not new and can be found in [18].

   The Apache Web Server and DBMS Cassandra were installed on the same server, which was called WebServer and was virtualized by Xen. The Hypervisor Xen was installed on two Xen Servers (XenServer 1 and 2) that were used for migration of the Web Server host. NFS is the server responsible for sharing virtual machines' images and virtual disks involved in the migration process. The existence of this element is one of the requirements of the Xen hypervisor [19]. The live migration process is highlighted in Figure 1.
2. *Hosts Settings:* The real machine used to install the virtual hosts in this phase had 16 GB RAM, Intel® Core™ i7-4790 CPU @ 3.60 GHz 64 bits and 680 GB of disk. This CPU was the same used at all virtualized hosts. Table 1 gives the virtual machine descriptions. All the machines (real and virtual) used on the experiment had Ubuntu 14.04 Desktop as the Operating System (OS), except the Web Server host, which had Ubuntu 14.04 Server.

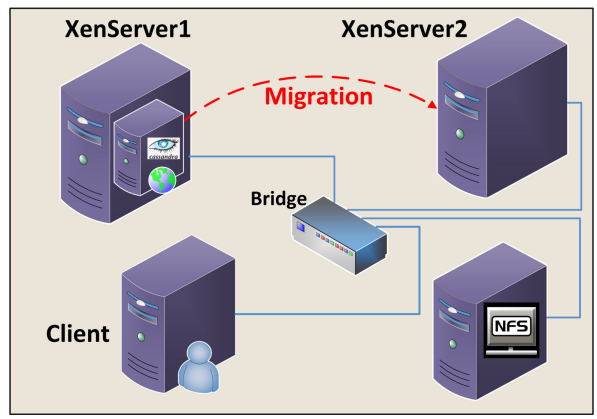


Figure 1. Environment totally with virtualized machines

[6] http://www.vmware.com/br/products/workstation

Hardware	XenServer1/XenServer2	WebServer	NFS	Client
Memory	6 GB	2 GB	4 GB	2 GB
Disc Size	40 GB	10 GB	25 GB	25 GB

Table 1. Virtual machines descriptions

B. Physical Servers

1. *The Environment:* it consisted of four real machines with one virtualized host that migrated between Xen Servers 1 and 2 using the same organization presented in Figure 1. The real hosts were: XenServer1, XenServer2, Client, the NFS and an Ethernet Switch. The Apache Web Server and DBMS Cassandra were installed on the same server, which was called Web Server and was virtualized by Xen. The Xen Hypervisor was installed on two Xen Servers (1 and 2) that were used for migration of the Web Server host. The real environment used the same machine's configurations as presented on virtual environment.

C. Experimental Design

Our experiments consisted of a $2^k r$ factorial design [20] to determine the effect of k factors, in our case $k = 2$ that are two factors each at two levels. We used $r = 10$, which means we made ten repetitions in each treatment. Following this design, it was made $2^k r = 2^2 10 = 40$ observations for each benchmark.

Table 2 gives the factor level combinations for each experiment made with benchmarks YCSB and *ab*. We considered two factors, Environment and Condition, each with two levels that were Real Machine or Virtual Machine in Environment factor and With Migration or Without Migration in Condition factor. These are our primary factors whose effects were quantified.

Factor	Level − 1	Level 1
Environment	Real Machine	Virtual Machine
Condition	With Migration	Without Migration

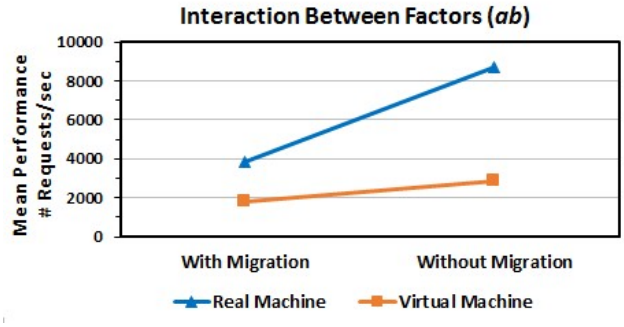
Table 2. Factors and levels of the design

We know there are secondary factors that impact the performance but such impacts were not considered in quantifying. For example, we were not interested in determining whether performance with *ab* is better than that of with YCSB.

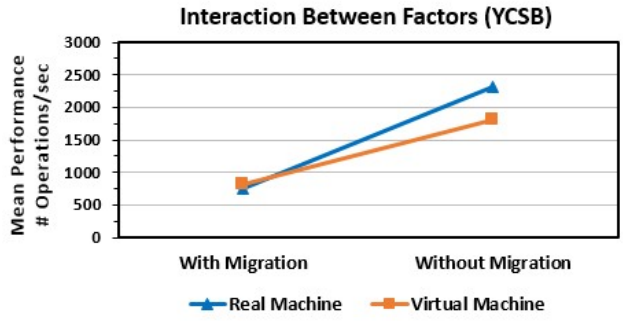
5 STATISTICAL ANALYSIS

To decide how to analyze the data collected, some statistical reviews were performed as follows. Two factors A and B are said to interact if the effect of one depends upon the level of the other one. Figures 2 a) and 2 b) give the interaction between Condition and Environment levels for *ab* and YCSB benchmarks. As shown in the

graph for *ab*, the lines of Real and Virtual Machines are not parallel, indicating an interaction between them [20]. The same can be observed from YCSB's graph.



a) *ab* benchmark

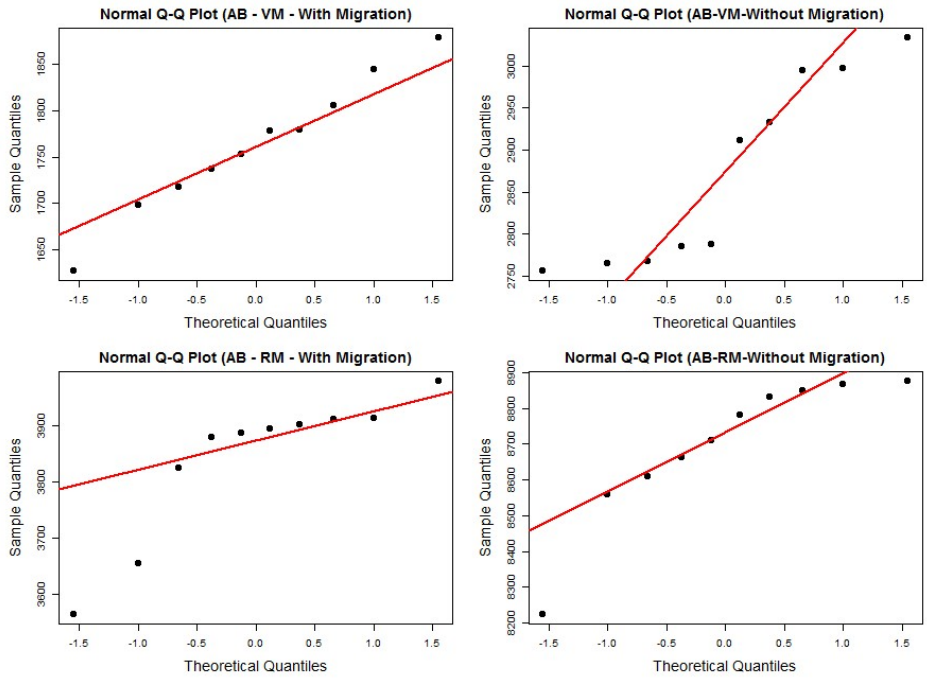


b) YCSB benchmark

Figure 2. Graphical presentation of interacting factors for a) *ab* benchmark and b) YCSB benchmark

Many statistical techniques assume the data are normally distributed, facilitating data analysis. Our data were evaluated aiming to verify whether data meets normal distribution. Some statistical tests were performed, as the Shapiro-Wilk test (whose parameter is W), and p-values less than 0.05 with $W < 1.0$ were found. In almost all results the null hypothesis was refuted (i.e., if p-value is less than the 0.05 significance level and $W < 1.0$, the null hypothesis is rejected). Only one test with a p-value = 0.9963 (> 0.05) and W very close to 1.0 ($W = 0.9896$) was that with *ab* on VM and with live migration. This result agrees with the chart on Figure 3 a) upper left where it is possible to see the most points fall along on a straight line.

As can be observed from all the others charts on Figure 3, few points fall along on a straight line. We conclude that the data samples collected come from a different distribution than normally. These results, based on graphs and in statistical test analysis lead to the decision to use non-parametric statistical tests. From the



a)

analysis made, we could conclude that the most appropriate test for these samples is the Mann-Whitney-Wilcoxon Test (or Wilcoxon rank sum test, or Mann-Whitney U-test), a non-parametric test by which we can decide whether the population distributions are identical not assuming they would follow the normal distribution [21].

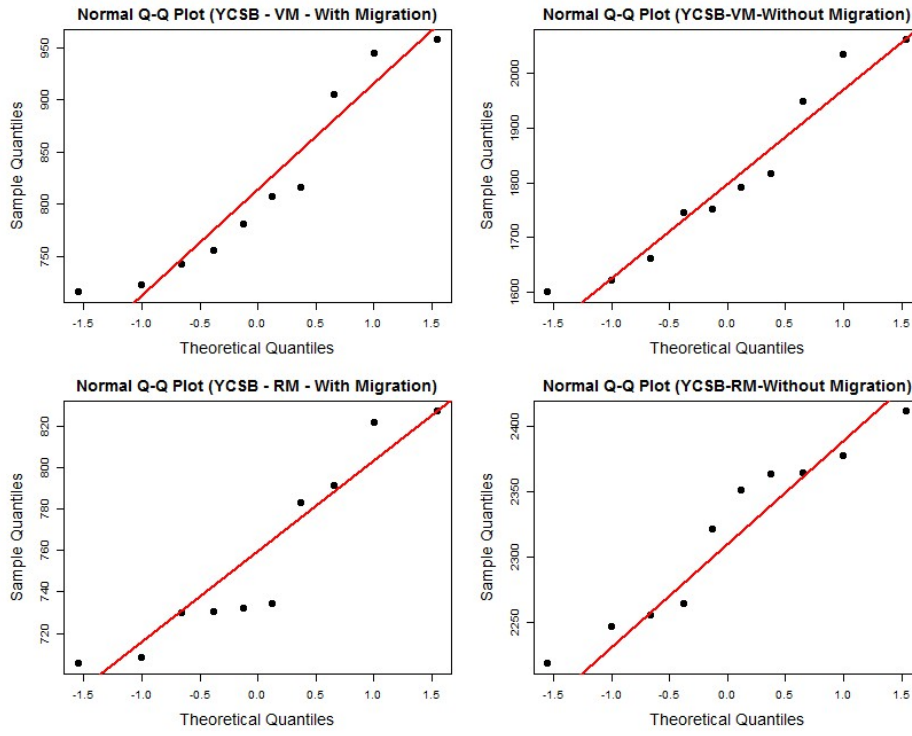
Many hypotheses could be tested with our results but, we believe the most important for statistical testing is to identify in which environment (real or virtual) a service deployed on VM provides a better performance during a migration event. This information is important to understand the client's perception of the service.

With this objective, the following hypotheses were tested:

- For the Web server tested through benchmark *ab*:

  H-I$_0$: The performance obtained by the benchmark *ab* on real environment, during a migration process, ***is equal to*** the performance obtained on a virtual environment.

  H-I$_1$: The performance obtained by the benchmark *ab* on the real environment, during a migration process, ***is greater than*** the performance obtained on a virtual environment.



b)

Figure 3. Normal quantile-quantile plot data for a) *ab* benchmark on Virtual and Real environments and b) YCSB benchmark on Virtual and Real environments

- For the DBMS server tested through benchmark YCSB:

  H-II$_0$: The performance obtained by the benchmark YCSB on real environment, during a migration process, **is equal to** the performance obtained on a virtual environment.

  H-II$_1$: The performance obtained by the benchmark YCSB on the real environment, during a migration process, **is greater than** the performance obtained on a virtual environment.

6 PERFORMANCE EVALUATION

To measure the impact, perceived by the host client, of the live migration of the applications running inside the virtual server, two benchmarks were used, one for each of the tested services (web server and DBMS). The benchmarks were running individually on the Client to test the web server and DBMS on remote Web Server

host, which was virtualized on Xen Server1. Measurements were made on servers both with and without migration. Benchmarks were configured as follows:

Apache benchmark (*ab*): The concurrency level was fixed in 10 parallel users using services. The number of multiple requests to the URL of the web server was fixed at 10 000 000. The maximum number of seconds to spend for benchmarking was fixed at 400 seconds. And 10 repetitions were made with these configurations. The response metric was the number of answered requests per second.

Yahoo! Cloud Serving Benchmark (YCSB): The number of client threads was fixed on 5, which indicates the amount of load offered against the database. The number of records to be used on the test was fixed at 100 000. The core workload used was a mix of 50 %/50 % of load and select and update operations. The number of operations to perform was fixed at 600 000. In the same way, 10 repetitions of the test with these configurations were performed. The evaluated metric was the number of operations per second.

A. Performance with Benchmark *ab*

A performance reduction is common for a system under migration, which is a function of time (i.e., conditions are different at each stage: pre-migration, mid-migration and post-migration). It is possible to see this pattern with data collected on client, with *ab* benchmark, during a migration event in Figure 4 (the same behavior was observed with YCSB's data). However, for the purpose of this work, it is desired to establish an idea of the impact caused in a service, running in VM, during a complete migration process. To this end, the temporal variations were discarded aiming the interpretation of the process as a whole.

Given a few factors that affect the system performance, it is important to know the effects of each factor individually [20].

Table 3 gives the factor level combinations for each experiment made with benchmark *ab*. The effects were quantified using the mean of ten repetitions by treatment.

From Table 3 we can see that, on average, the virtual machine processing capacity in number of requests/seconds was equivalent to 33 % of the capacity reached by real machines in a non-migration environment. The results obtained with migration show that the virtual machines had 46 % of the capacity when compared with the real machines.

Environment	With Migration	Without Migration
Real Machine	3 840.81	8 697.43
Virtual Machine	1 761.77	2 873.10

Table 3. Performance in number of answered requests/sec with *ab*

Figure 5 demonstrates the performance of the *ab* benchmark running on virtual and real environments. The lines above each bar in the bar graphs are

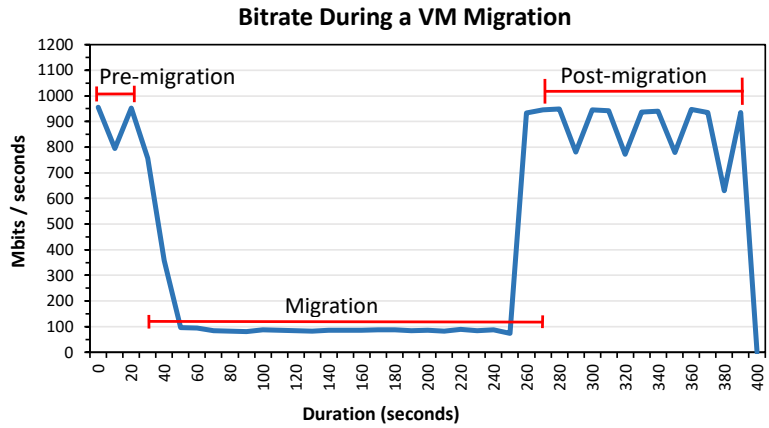


Figure 4. Throughput during a complete *ab* benchmark run on real environment

the standard errors bars. As shown in the graph, the performance of the web server was better running without migration than when a migration occurs as expected.

Comparing the individual means on the virtual environment, the results with migration reached the equivalent of 61 % of the number of requests/seconds obtained by the experiments without migration, indicating a considerable reduction in processing capacity because of the migration process.

When analyzing the results of *ab* on the real environment (Figure 5), the enhancer of performance on real machine when comparing with the results on VM is clear. Again, the performance without migration exceeded the one with live migration in all ten rounds, with a larger contrast between the conditions than that observed on VM.

In the real machine environment, experiments with migration only obtained 44 % of the processing capacity of the environment without migration. The overhead caused by the imposition of a virtualization layer in the computing environment as well as the live migration process becomes evident. This simply reflects the services offered by virtualized servers. This fact was also noticed in testing with DBMS.

Considering all the comparisons, it is possible to check that the impact of migration on the real environment was high, however, lower than on the virtual environment, since in the former we had a reduction of 44 % of the capacity without migration, while with the virtual machines a reduction of 61 % of the capacity can be observed when comparing the environment with and without migration.

Based on these results, we want to highlight that the decision makers should have a better understanding of the impact on the services offered by virtual

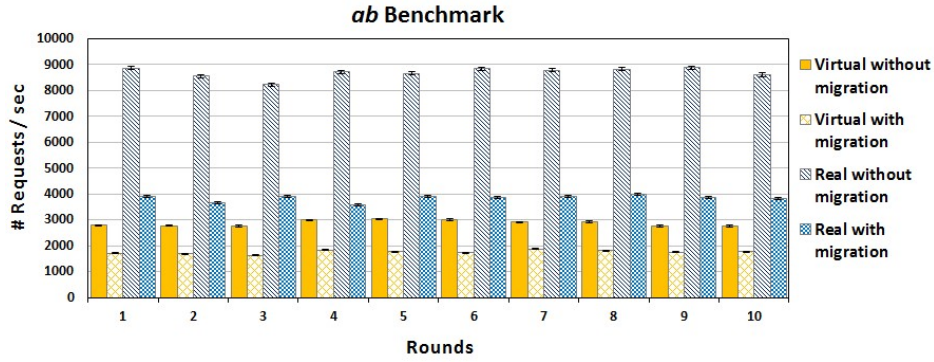


Figure 5. Performance of *ab* on virtual and real environments (higher is better)

servers. In this way, they can weigh the pros and cons and decide more safely whether to move or not their services to Cloud providers.

B. Performance with Benchmark YCSB

Table 4 gives the factor level combinations for each experiment made with benchmark YCSB. As with *ab*, the effects were quantified taken the mean of ten repetitions by treatment.

The results of the ten rounds on virtual and real environments with YCSB can be seen in Figure 6. As with the results of *ab*, YCSB on VM also prevailed in all rounds when running without migration, when compared with the environment migrating the server. Analyzing the individual means on the virtual environment, the results with migration reached the equivalent of 45 % of the performance of the ones without migration in number of operations/second.

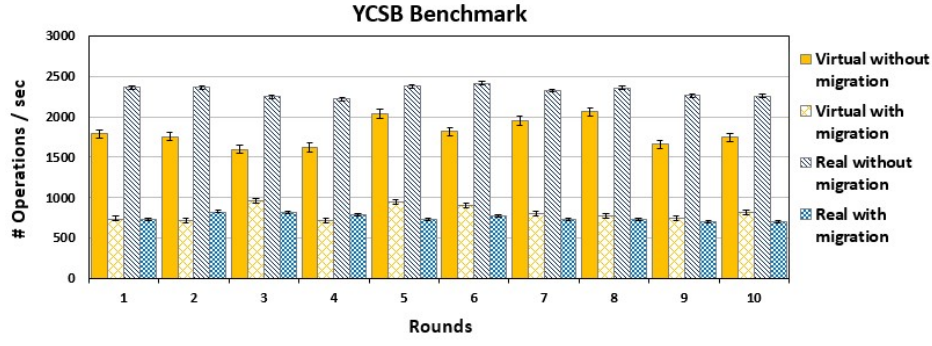


Figure 6. Performance of YCSB on virtual and real environments (higher is better)

When analyzing the performance of the YCSB, with and without migration, on the real environment, the results show that the tests without migration also

Environment	With Migration	Without Migration
Real Machine	756.26	2 316.85
Virtual Machine	814.49	1 802.68

Table 4. Performance in operations/sec with YCSB

overcame those with live migration with a significant difference, since migration condition obtained only 32 % of the performance from the non-migrating environment.

A very interesting result from YCSB evaluation is that, when migrating the machines, the virtual environment obtained better results than the real one, as presented in Table 4. Analyzing the migration logs, it was possible to see that the number of pages transferred during migration was equivalent in both environments (about to 1.4 million pages), but transferring rate was higher on the virtual environment than on the real environment. Since a considerable amount of information from the virtual machines remains only in memory, to reallocate this memory to another process in the same machines was faster than to transmit this information over the network. Because of it, the benchmark started to run without migration overhead faster on the virtual environment than on the real one (approximately 2 minutes), what was not noticed in *ab* benchmark, since it obtained the same migration time in both environments.

Also, performance without migration on real environment was much closer to the virtual environment than the result obtained using the Web Server. Using DB, the virtual environment had 78 % of the performance of the real environment, while using Web Server, the virtual machine obtained only 33 % of the capacity when running without migration. It is possible to note with these results that DB service (which demands more processing capacity) has been accommodated much better to the virtual environment and to migration than Web service (demanding more access to stored information).

To test the previously made null-hypotheses we applied the Mann-Whitney-Wilcoxon U-test, using the R[7] statistical software, all tests used a significance level of 0.05. The results were as follows:

For the Web server tested through benchmark *ab*: the p-value turns out to be 5.413e−06, and is less than the 0.05 significance level, we reject the null hypothesis $H\text{-}I_0$ and accept the alternative $H\text{-}I_1$. *That is, the performance obtained by the benchmark ab on the real environment, during a migration process,* ***is superior to*** *the performance obtained on the virtual environment.*

For the DBMS server tested through benchmark YCSB: the p-value turns out to be 0.9173, greater than the 0.05 significance level, in this case,

[7] `https://www.r-project.org/`

we can accept the hypothesis $H\text{-}II_0$ of statistical equality of the means of two groups. *That is, the performance obtained by the benchmark YCSB on the real environment, during a migration process,* ***is equal to*** *the performance obtained on the virtual environment.*

C. Allocation of Variation

As the data from the experiments suggests, there is a difference between the treatments and the interaction between factors, as shown in Figure 2. We would like to confirm those expectations. To do so, box plots were made (*ab* and YCSB benchmarks on virtual and real machine environments), in Figures 7 a) and 7 b) we have the results for *ab*, and in Figures 8 a) and 8 b) we have the results for YCSB. As can be seen, the graphs suggest differences between the treatments used. Then, it was decided to measure the allocation of variation. The percentage of variation explained by each factor is helpful in deciding whether a factor has a significant impact on the response [20]. The factors which explain a high percentage of variation are considered important.

With the factors and levels of Table 2, following the $2^k r$ factorial design described in [20], let us define two variables x_A and x_B as follows:

$$x_A = \begin{cases} -1, & \text{with migration,} \\ 1, & \text{without migration,} \end{cases} \tag{1}$$

$$x_B = \begin{cases} -1, & \text{real machine,} \\ 1, & \text{virtual machine.} \end{cases} \tag{2}$$

The performance y in number of requests/second (for *ab*) or operations/second (for YCSB) can now be regressed on x_A and x_B using a nonlinear regression model of the form:

$$y = q_0 + q_A x_A + q_B x_B + q_{AB} x_A x_B + e. \tag{3}$$

The terms in (3) are: y is mean performance; x_A is the effect of condition; x_B is the effect of the environment; x_{AB} is the effect of interactions between environment and condition; q_0, q_A, q_B and q_{AB} are the effects; and e is the experimental error.

First, we used the sign table to analyze our $2^k r$ factorial design and compute the effects, as described in [20].

1. *Allocation of Variation for ab*: following (3) and with the data from Table 3 through the sign table, the model (4) of mean performance from *ab* was developed:

$$y = 4\,293.28 + 1\,491.99 x_A - 1\,975.84 x_B - 936.32 x_A x_B + e. \tag{4}$$

Thus, making the calculations, we found that the total variation can be divided into four parts. Factor B (environment) explains 55.58 % of the variation. Factor A (condition) explains 31.69 % of the variation and interaction AB explains 12.48 % of the variation. The remaining 0.24 % is unexplained and attributed to errors.

From these *ab* results, we can conclude that the environment had more impact in *ab* tests than the condition (using VM live migration or not).

2. *Allocation of Variation for YCSB*: following (3) and with the data from Table 4 through the sign table, the model (5) of mean performance from YCSB was developed:

$$y = 1\,422.57 + 637.19x_A - 113.98x_B - 143.10x_Ax_B + e. \tag{5}$$

Thus, making the calculations, we found that the Factor A (condition) explains 90.46 % of the variation. Factor B (environment) explains 2.89 % of the variation, it can be ignored, and interaction AB explains 4.56 % of the variation. The remaining 2.08 % is unexplained and is attributed to errors.

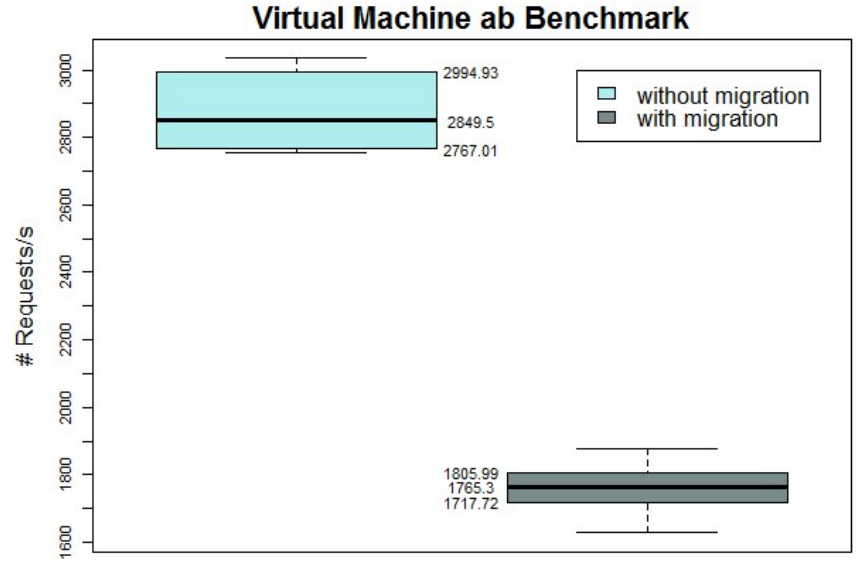
From the YCSB results, we can conclude that the condition had almost all impact in YCSB tests. Use the DB on the real or virtual machine makes no difference in our tests since each environment had an advantage in one of the tests.

In deriving the expressions for effects, we made some assumptions. These assumptions lead to the observations being independent and normally distributed with constant variance. To verify these assumptions, that were made with the regression model, it was decided to use visual tests. To do that, Figures 9 a) and 9 b) give a plot of residuals and a normal quantile-quantile plot for *ab*. As there is no trend in Figure 9 a), we can assume the errors are independently and identically distributed. In Figure 9 b) the residuals appear to be approximately normally distributed. Thus, the model appears to be valid for our experiment with *ab*.

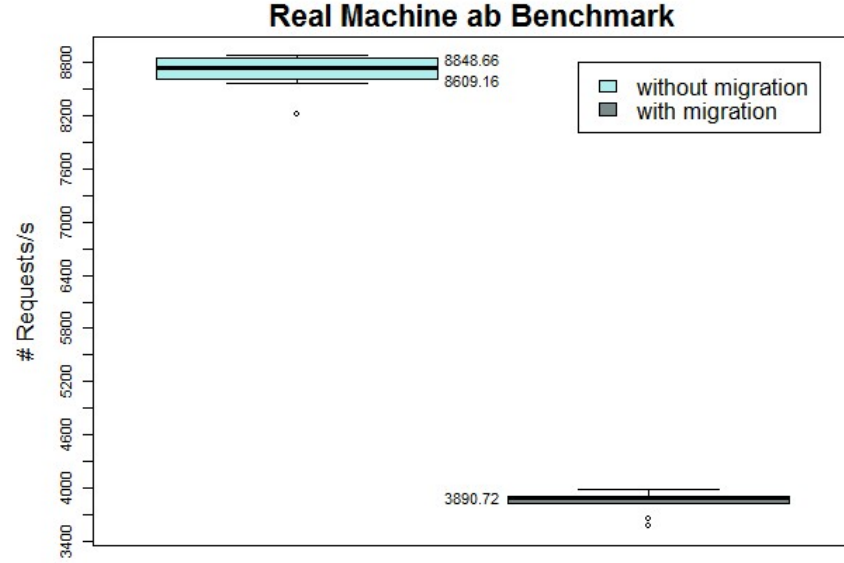
The same analysis made for *ab* can be applied to YCSB, using Figures 10 a) and 10 b). We can assume the errors are independently and identically distributed, and the residuals appear to be approximately normally distributed as well. Thereby, the model also seems to be valid for our experiment with YCSB.

D. Confidence Intervals for the Effects

As seen in Section 4, the errors were normally distributed and, as calculated, this distribution has zero mean, it was possible to discover the confidence intervals for the effects. This information makes possible to find if the effects are significant. The standard deviation of errors can be estimated from the sum of

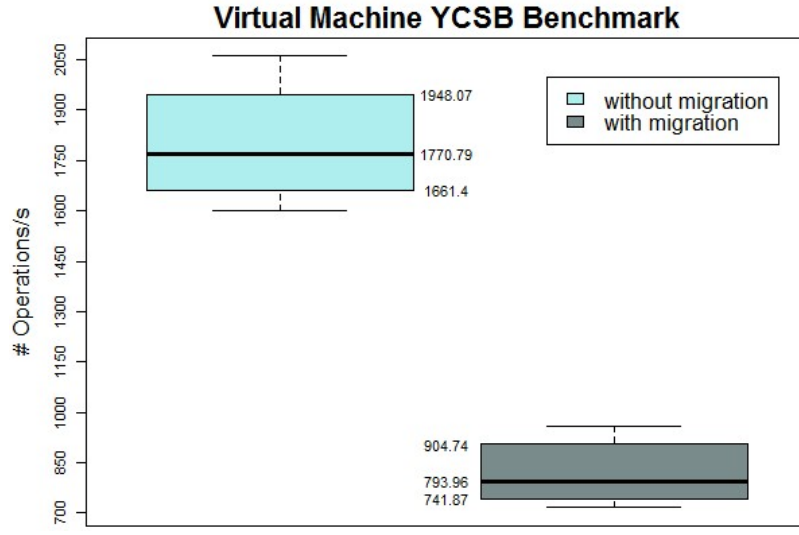


a)

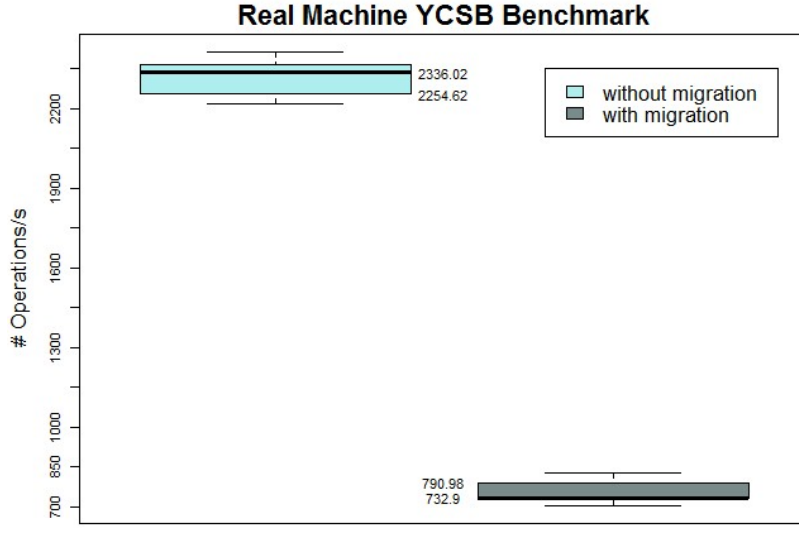


b)

Figure 7. Box plot of *ab* performance on a) VM and b) Real Machine (higher is better)



a)



b)

Figure 8. Box plot of YCSB performance on a) VM and b) Real Machine (higher is better)

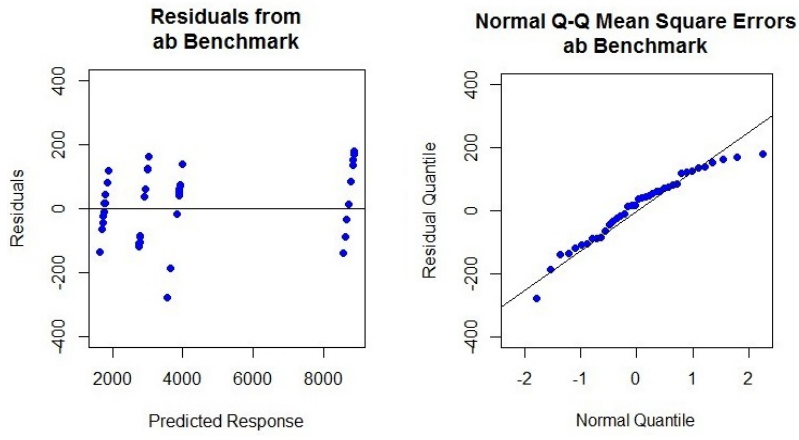


Figure 9. Plot of a) residuals versus predicted response and b) normal quantile-quantile for *ab*

the squared errors (SSE) as follows:

$$S_e = \sqrt{\frac{SSE}{2^2(r-1)}} \tag{6}$$

and the standard deviation of effects is

$$S_{q_i} = \frac{S_e}{\sqrt{2^2 r}}. \tag{7}$$

Then, the confidence intervals for the effects is

$$q_i \mp t_{\left[\frac{1-\alpha}{2};2^2(r-1)\right]} S_{q_i}. \tag{8}$$

The t-value is read at $2^2(r-1)$ degrees of freedom, for our data the degrees of freedom is 36, and it was used 90 % of confidence interval. So, the t-value at 36 degrees of freedom and 90 % of confidence is 1.6883.

Table 5 has the Confidence Intervals (CI) for data from benchmarks *ab* and YCSB for q_0, q_A, q_B and q_{AB}.

As can be seen from Table 5, none of the intervals include a zero, therefore, all the effects are significant.

E. Performance Impact with Multiple Migrations

Considering the initial evaluations, it is possible to see that the impact of the migration in virtualized environments was less representative than in real envi-

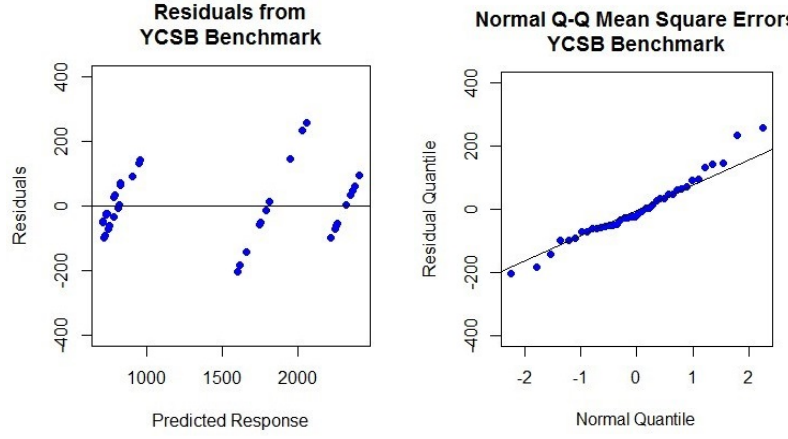


Figure 10. Plot of a) residuals versus predicted response and b) normal quantile-quantile for YCSB

Effects	**CI for *ab***	**CI for YCSB**
q_0	$(4\,256.769, 4\,329.785)$	$(1\,395.356, 1\,449.780)$
q_A	$(1\,455.481, 1\,528.496)$	$(609.982, 664.406)$
q_B	$(-2\,012.349, -1\,939.334)$	$(-141.194, -86.771)$
q_{AB}	$(-972.831, -899.815)$	$(-170.314, -115.890)$

Table 5. Confidence intervals for effects from *ab* and YCSB

ronments, even the virtualized services presented lower performance. Services virtualization is one of the key points in the development of many new computing paradigms during the last years.

One of them that can be highlighted is cloud computing, where thousands of services can be deployed in a shared infrastructure using Virtual Machines.

In cloud computing systems, it is usual to find the virtualized server migrating at the same time, arriving or leaving a host server. Therefore, we decided to evaluate the impact on service's performance caused when multiple migrations are made. We wanted to know, for instance, what is the Web Server performance on one VM that is migrating from a host when there is another VM arriving at the same time in the same host. Those tests were made only on the virtual environment, implemented on a server with 16 GB of memory and 500 GB of disk.

Figure 11 shows the architecture of this environment, indicating the fundamental elements for the migration process to be executed. The VM Client is responsible for generating the requests to the Apache Web server through

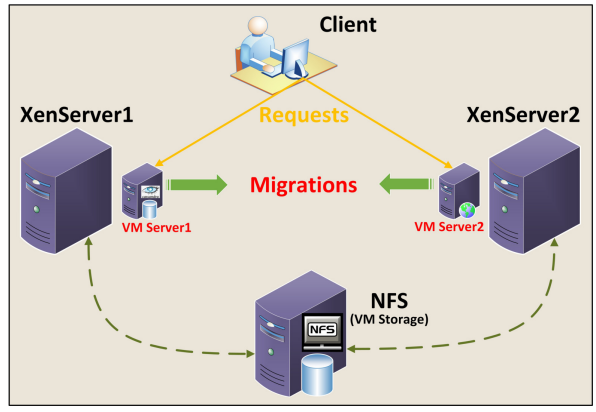


Figure 11. Environment totally with virtualized machines and with simultaneous migrations

the *ab* benchmark, as well as performing the transactions for the Cassandra database through the YCSB benchmark. It was from this machine that the benchmarks generated the triggered workload for Server1 and Server2, these two virtual servers were virtualized by Xen. The Hypervisor Xen was installed on two Xen Servers (XenServer 1 and 2) that were used for migration of the Server1 and Server2, as source and destination machines. They have the same memory, disk, and operating system settings as can be seen in Table 6.

The Server1 and Server2 VMs are hosting the Apache Web Server, as well as the DBMS Cassandra server. During the migration process, these were the servers that were migrated. The NFS was used the same way as described in previous sections.

Table 6 shows the hardware and software configurations of all VMs involved in the migration process. In all hosts, the OS used was Ubuntu 14.04.

VMs	**Memory (GB)**	**HD (GB)**
Client	2	25
XenServer1	4	40
XenServer2	4	40
Server1	2	10
Server2	2	10
Storage NFS	1	25

Table 6. Virtual machines descriptions

In the experiments, two virtual servers (Server1 and Server2) were used, one with Apache Web Server and another with Cassandra DB. Server1 was initially virtualized on XenServer1 and Server2 on XenServer2, then Server1 was migrated

to XenServer2 and, at the same time, Server2 was migrated to XenServer1. Measurements were made on these virtual servers with only one migration, simultaneous migrations, and without migration. Benchmarks *ab* and YCSB used the same configurations of the initial experiment and 10 repetitions of each test were performed to guarantee statistical representativeness.

Here, we call *"with migration"* when just one VM is migrating and receiving requests from a client; and we call *"with simultaneous migration"* when the two virtual servers (Web Server and DBMS) are migrating simultaneously, and both are receiving requests from clients. The results of these simulations can be seen in Figures 12 and 13.

Analyzing the requests per second metric, Figure 12 shows that, on average, the execution of *ab* with simultaneous migration resulted in a reduction of 31.3 % when compared to the non-migration approach, and 14.6 % when compared to a single migration. In this way, it is observed that for this case the simultaneous migration was the one that resulted in a greater impact on the performance of the service.

As can be seen in Figure 13, the data collected shows the migration performance implied, on average, an approximate reduction of 61.8 % in the YCSB throughput. However, it is also possible to note that the approaches with a single and with simultaneous migrations presented approximate values, that is, a simultaneous migration execution scenario implies, in average, a reduction equivalent to the single migration approach for this service.

As a conclusion from these results, it was possible to confirm with YCSB's data that the migration process imposes a significant overhead compared to non-migration tests on the virtualized environment. In addition, the occurrence of single migration or multiple migrations apparently did not imply significant differences, which should be confirmed by statistical tests. Regarding the tests with the *ab* benchmark, it was also possible to observe the migration impact in the Web service, however, not as significantly as with the DBMS Cassandra. If we observe the events with simultaneous migrations, we can see that this situation further degrades the Web service, reaching a 31.3 % drop in performance. These results confirm the results of the initial experiment, which indicates that non-relational database service was less affected by migration process than web content service.

To confirm the results obtained in the tests with simultaneous migrations, we performed a series of statistical tests of Mann-Whitney-Wilcoxon U-test, all with a significance level of 0.05, using statistical software R. The results were:

- With respect to *ab* benchmark, comparing the data from the tests with migration and with simultaneous migration, the p-value turns out to be 1.083e−05, less than the 0.05 significance level, in this case, we reject the null hypothesis of statistical equality of the means of the two groups. *That is, the performance obtained by the ab benchmark with only one server migrating*

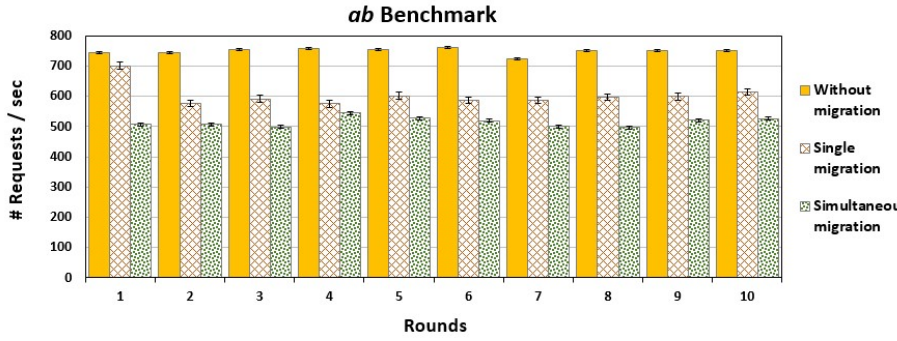


Figure 12. Performance of *ab* benchmark on virtual environment (higher is better)

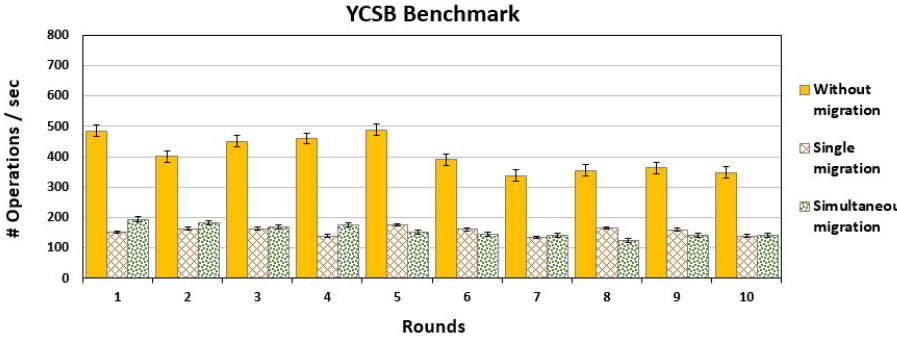
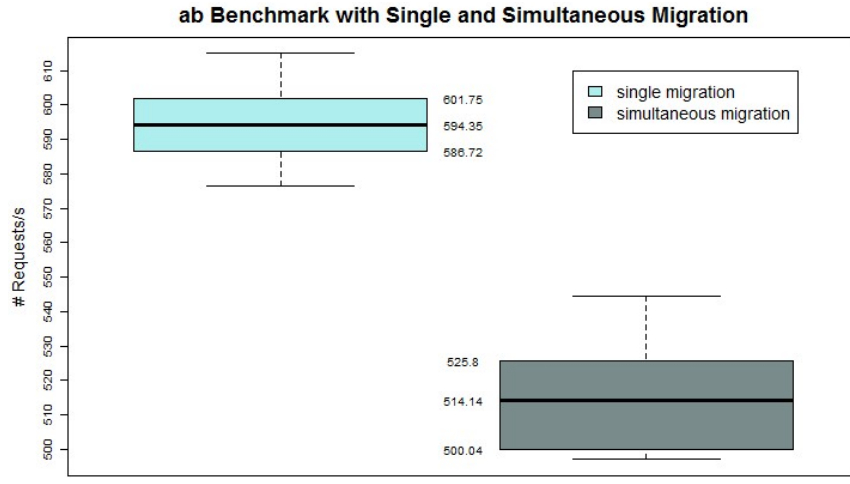


Figure 13. Performance of YCSB benchmark on virtual environment (higher is better)

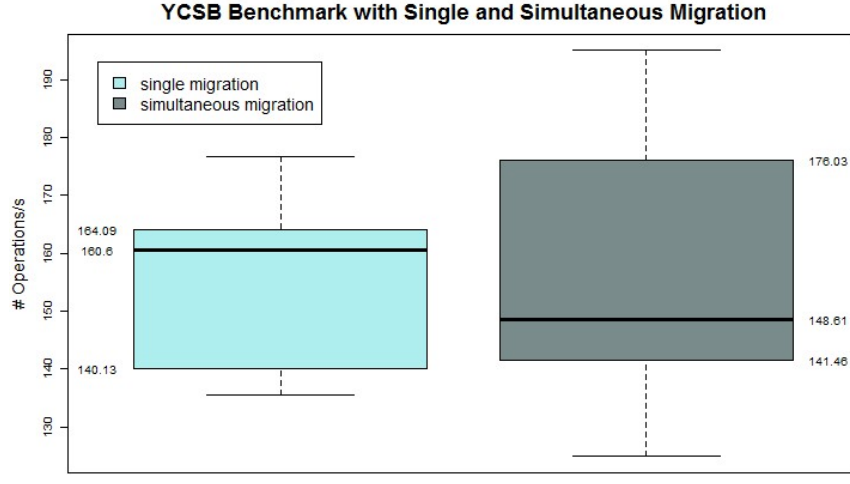
is different from *the performance obtained when simultaneous migrations were occurring.*

- The YCSB chart (Figure 13) indicated that the occurrence of single migration or multiple migrations did not imply, apparently, significant differences in the performances of the Cassandra DB. To test this hypothesis, the same statistical test was performed comparing the data of single migration and multiple migrations from YCSB benchmark. As the p-value turns out to be 0.8534, greater than the 0.05 significance level, in this case, we can accept the null hypothesis of statistical equality of the means of the two groups. *That is, it is not possible to statistically affirm, with a significance level of* 5 %*, that there are differences between the performance obtained by the YCSB benchmark when only one server is migrating and when multiple migrations are taking place.*

The results obtained with the statistical tests were confirmed with the box plot charts available in Figures 14 a) and 14 b). The difference between sim-



a)



b)

Figure 14. Box plot of the performance with single and with simultaneous migration of a) *ab* and b) YCSB (higher is better)

ple migration and multiple migrations is evident from *ab* benchmark data. Whereas, for the YCSB, it is not possible to state that there is a statistical difference.

7 CONCLUSION AND FUTURE WORK

Experiments were conducted in fully virtual and real environments migrating and not migrating the content servers. As achievements, we can highlight:

- The overhead caused by live migration process is significant, noting that on both benchmarks *ab* as YCSB, the number of operations or requests per second were significantly reduced.
- The results showed that the *ab* benchmark had superior performance when running on a fully real environment, regardless of the scenario being with or without live migration. This is because the benchmark *ab* makes much use of the network interface which eventually becomes a bottleneck for operations, since the performance of network resources with virtual machines is reduced.
- Comparing the results of the YCSB benchmark in a fully virtualized environment with real, it was observed that the fully virtualized presented lower performance without migration scenario, but exceeded the real environment when there was live migration. Here there is the intense transfer bottleneck of data between hosts since the YCSB performs operations on the host database server. Therefore, the performance without live migration was higher in the real machine. In the case of migration, we have the Ethernet bridge factor that, in the virtual scenario, favored runtime since all hosts and the bridge were on the same real machine.
- When multiple migrations were conducted in the environment two different behaviors could be observed according to the service used. When web service was in place there was a significant reduction in performance. However, running Cassandra no additional reduction was observed.

In summary, we have identified that the overhead caused by the virtual machine live migration process observed from the client's point of view is very impactful, since performance was degraded in all results of the benchmark execution on real environment when compared to virtual environment.

As future work, we are going to plan to deploy different services to check the impact perceived in each one, to extend the experiments with multiple migrations and to use a heterogeneous environment where we would find virtual and real servers used as hosts for migration process. It is also intended to include KVM as a hypervisor, as well as measuring the impact of network infrastructure on results.

REFERENCES

[1] Ramachandran, M.—Chang, V.: Towards Performance Evaluation of Cloud Service Providers for Cloud Data Security. International Journal of Information Management, Vol. 36, 2016, No. 4, pp. 618–625, doi: 10.1016/j.ijinfomgt.2016.03.005.

[2] Kim, A.—Lee, J.—Kim, M.: Resource Management Model Based on Cloud Computing Environment. International Journal of Distributed Sensor Networks, Vol. 12, 2016, doi: 10.1177/1550147716676554.

[3] Chen, W.—Shang, Z.—Tian, X.—Li, H.: Dynamic Server Cluster Load Balancing in Virtualization Environment with OpenFlow. International Journal of Distributed Sensor Networks, Vol. 11, 2015, No. 7, doi: 10.1155/2015/531538.

[4] Xing, G.—Xu, X.—Xiang, H.—Xue, S.—Ji, S.—Yang, J.: Fair Energy-Efficient Virtual Machine Scheduling for Internet of Things Applications in Cloud Environment. International Journal of Distributed Sensor Networks, Vol. 13, 2017, doi: 10.1177/1550147717694890.

[5] Zoraja, I.—Trlin, G.—Sunderam, V.: Eliciting the End-to-End Behavior of SOA Applications in Clouds. Computing and Informatics, Vol. 35, 2016, No. 2, pp. 259–281.

[6] Marshall, D.: Understanding Full Virtualization, Paravirtualization, and Hardware Assist. VMware White Paper, 2007.

[7] Magalhães, D. V.—Soares, J. M.—Gomes, D. G.: Análise do Impacto de Migração de Máquinas Virtuais em Ambiente Computacional Virtualizado. XXIX SBRC, Mato Grosso do Sul, Brazil, 2011, pp. 235–248.

[8] Bobák, M.—Hluchý, L.—Tran, V. D.: Application Performance Optimization in Multicloud Environment. Computing and Informatics, Vol. 35, 2016, No. 6, pp. 1359–1385.

[9] Padala, P.—Zhu, X.—Wang, Z.—Singhal, S.—Shin, K. G.: Performance Evaluation of Virtualization Technologies for Server Consolidation. HP Labs Technical Report HPL-2007-59R1, April 11, 2007.

[10] Li, J.—Wang, Q.—Jayasinghe, D.—Park, J.—Zhu, T.—Pu, C.: Performance Overhead among Three Hypervisors: An Experimental Study Using Hadoop Benchmarks. IEEE International Congress on Big Data, Santa Clara, USA, 2013, pp. 9–16, doi: 10.1109/BigData.Congress.2013.11.

[11] Sahoo, J.—Mohapatra, S.—Lath, R.: Virtualization: A Survey on Concepts, Taxonomy and Associated Security Issues. 2010 Second International Conference on Computer and Network Technology (ICCNT), IEEE, Bangkok, Thailand, 2010, pp. 222–226, doi: 10.1109/ICCNT.2010.49.

[12] XenSource: A Performance Comparison of Commercial Hypervisors. XenEnterprise vs. ESX Benchmark Results, XenSource, 2007.

[13] VMware: A Performance Comparison of Hypervisors. VMware White Paper. `https://www.vmware.com/pdf/hypervisor_performance.pdf`, 2007, accessed 10 October 2016.

[14] Reddy, P. V. V.—Rajamani, L.: Performance Evaluation of Hypervisors in the Private Cloud Based on System Information Using SIGAR Framework and for System

Workloads Using Passmark. International Journal of Advanced Science and Technology, Vol. 70, 2014, pp. 17–32.

[15] Martins, G.—Bezerra, P.—Gomes, R.—Albuquerque, F.—Costa, A.: Evaluating Performance Degradation in NoSQL Databases Generated by Virtualization. 2015 Latin American Network Operations and Management Symposium (LANOMS), João Pessoa, Brazil, 2015, pp. 84–91, doi: 10.1109/LANOMS.2015.7332675.

[16] Galloway, M.—Loewen, G.—Vrbsky, S.: Performance Metrics of Virtual Machine Live Migration. 2015 IEEE 8th International Conference on Cloud Computing, New York, USA, 2015, pp. 637–644, doi: 10.1109/CLOUD.2015.90.

[17] Isci, C.—Liu, J.—Abali, B.—Kephart, J. O.—Kouloheris, J.: Improving Server Utilization Using Fast Virtual Machine Migration. IBM Journal of Research and Development, Vol. 55, 2011, No. 6, 12 pp., doi: 10.1147/JRD.2011.2167775.

[18] Zhang, F.—Chen, J.—Chen, H.—Zang, B.: CloudVisor: Retrofitting Protection of Virtual Machines in Multi-Tenant Cloud with Nested Virtualization. Proceedings of the Twenty-Third ACM Symposium on Operating Systems Principles (SOSP '11), Cascais, Portugal, 2011, pp. 203–216, doi: 10.1145/2043556.2043576.

[19] Wood, T.—Shenoy, P.—Venkataramani, A. et al.: Sandpiper: Black-Box and Gray-Box Resource Management for Virtual Machines. Computer Networks, Vol. 53, 2009, No. 17, pp. 2923–2938, doi: 10.1016/j.comnet.2009.04.014.

[20] Jain, R.: The Art of Computer Systems Performance Analysis: Techniques for Experimental Design, Measurement, Simulation, and Modeling. 1st edition. John Wiley and Sons, 1990.

[21] Hollander, M.—Wolfe, D. A.—Chicken, E.: Nonparametric Statistical Methods. 3rd edition. John Wiley and Sons, 2013.

Petrônio Carlos Bezerra graduated in computer science at the Federal University of Paraíba, Campina Grande, Brazil in 1999. Currently, he is Professor at the Federal Institute of Education, Science and Technology of Paraiba, Campina Grande, Brazil. He is a doctoral student at Federal University of Campina Grande. His current research interests include computer networks, software-defined networks and virtualization.

Marcela Tassyany Galdino Santos is the undergraduate student at the Federal Institute of Education, Science and Technology of Paraiba, Campina Grande, Brazil. Her current research interests include computer networks, software-defined networks, virtualization, cloud computing and optical communications.

Edlane de Oliveira G. Alves is the undergraduate student in telematics at the Federal Institute of Education, Science and Technology of Paraíba, Campina Grande Campus. She has participated in research focusing on optical communications, virtualization, technology and production with an emphasis on open source academic control systems. She is interested in cloud computing, computer networks and currently participating in projects in the area of software-defined networks, virtualization and digital inclusion with an emphasis on education.

Fellype Albuquerque graduated in computer science at the Federal University of Campina Grande, Campina Grande, Brazil with the emphasis on computer networks and cloud computing. Currently he is developing projects in the area of cloud computing in the Distributed Systems Laboratory.

Gustavo Nóbrega Martins received his Master degree in computer science from the Federal University of Campina Grande and graduation degree in computer science from the State University of Paraíba. Currently, he is the doctoral student at the Federal University of Campina Grande, Campina Grande, Brazil. His research interests include wireless sensor network, software defined radio, computer networks and virtualization technology.

Reinaldo Gomes has been Assistant Professor of the Computing and Systems Department of UFCG since 2010. Degree in telematics technology from the Federal Institute of Education, Science and Technology of Paraiba in 2004, Master in computer science from the Federal University of Pernambuco in 2005 and Ph.D. in computer science from the Federal University of Pernambuco in 2010. Since 2004 he has been actively involved in research and development projects with national and international cooperation, concerning development of communication protocols, traffic evaluation, wireless communication technologies and advanced applications for future internet.

Anderson Fabiano B. F. da Costa graduated in telematics at the Federal Institute of Education, Science and Technology in 2004, he received his Master and Doctor degrees from the Federal University of Pernambuco in 2005, and in 2011. Currently, he is Professor at the Federal Institute of Education, Science and Technology of Paraiba, Campina Grande, Brazil. He has experience in computer science, with emphasis on computer networks, acting in the following themes: software-defined networks, virtualization, wireless networks and cluster analysis.

Computing and Informatics, Vol. 38, 2019, 321–342, doi: 10.31577/cai_2019_2_321

ADAPTIVE REVERSIBLE DATA HIDING SCHEME FOR DIGITAL IMAGES BASED ON HISTOGRAM SHIFTING

Sonal Kukreja, Geeta Kasana, Singara Singh Kasana

Computer Science and Engineering Department
Thapar Institute of Engineering and Technology
Patiala, Punjab, India-147004
e-mail: kukreja.sonal@gmail.com, {gkasana, singara}@thapar.edu

Abstract. Existing histogram based reversible data hiding schemes use only absolute difference values between the neighboring pixels of a cover image. In these schemes, maxima and minima points at maximum distance are selected in all the blocks of the image which causes shifting of the large number of pixels to embed the secret data. This shifting produces more degradation in the visual quality of the marked image. In this work, the cover image is segmented into blocks, which are classified further into complex and smooth blocks using a threshold value. This threshold value is optimized using firefly algorithm. Simple difference values between the neighboring pixels of complex blocks have been utilized to embed the secret data bits. The closest maxima and minima points in the histogram of the difference blocks are selected so that number of shifted pixels get reduced, which further reduces the distortion in the marked image. Experimental results prove that the proposed scheme has better performance as compared to the existing schemes. The scheme shows minimum distortion and large embedding capacity. Novelty of work is the usage of negative difference values of complex blocks for secret data embedding with the minimal number of pixel shifting.

Keywords: Reversible data hiding, complexity, MSE, PSNR, histogram shifting, BPCS, firefly

1 INTRODUCTION

With the growth in multimedia communication, security of digital data being transferred and stored has been an important concern. Data hiding is one of the approaches used for security of digital data. It allows the sender to embed the information into digital contents like text, images, audio and video. The digital object in which the information is embedded is termed as cover object and the information to be embedded is termed as secret data. The cover object after embedding the secret data is known as marked object. In a Reversible Data Hiding (*RDH*) the secret data is embedded inside some cover media in such a way that it should not be visible with the human eyes. At the receiver side, the scheme should have the ability to restore the cover media as well as the secret data without any distortion [1, 14]. Specific research has been done on the sensitive images like medical images, satellite images, etc. [22].

Histogram modification based schemes are a remarkable contribution in *RDH* schemes. In these approaches, the statistical properties of the image are used to embed the secret data. Ni et al. [14] proposed the *RDH* scheme, in which the zero and peak points of a cover image histogram are recorded. All the pixel values lying in between these zero and peak values are shifted by one to create the empty spaces. These spaces are then used to embed the secret data bits. Peak Signal to Noise Ratio (*PSNR*) of this scheme is around 48 dB with a good embedding capacity. A modification of this scheme was proposed by the same author and it was used as an authentication scheme for semi fragile images [15]. This scheme was further improved by Fallahpour et al. [3] who proposed that instead of hiding the data in the entire image, the image can be divided into blocks and then the same histogram shifting using the peak and the zero points can be repeated for all the blocks, thereby enhancing the embedding capacity. This embedding capacity was further enhanced by Tsai et al. [23] by embedding the secret data in the residual images instead of image histograms. The pixels within the image were classified as wall pixels and non wall pixels, and then the interpolation error of the wall pixels, and the difference values of the non wall pixels and the parent pixels were used for data embedding. The embedding capacity was further enhanced by exploring the prediction values in a rhombus prediction scheme in Chang et al. [2]. Another variation in the histogram based approach was proposed by Wang et al. [24] by introducing the concept of location maps and manipulating the maximum frequency values of the histogram assuming their intensity values and updating these peak values with the other pixel value of the same segment. Hong et al. [7] enhanced the embedding capacity by using the dual binary trees instead of shifting the difference values. To construct a sharper histogram, Lin et al. [11] utilized difference image histograms. The correlation of two adjacent pixels is considered in this scheme. In [10], the host image is divided into sub-images by sampling. One of these sub-images is selected as a reference image to compute differences with others and the secret message is embedded by multilevel histogram modification. Luo et al. [12] improved this scheme by selecting the median of every block to construct a reference image, which leads to a sharper

histogram. In Pan et al. scheme [18], histogram is constructed for every block, and the peak point is selected as the reference pixel to compute differences with other pixels.

1.1 Motivation for the Proposed Work

From the literature survey, it is concluded that most of the existing histogram shifting based *RDH* schemes [3, 14, 11] have not considered the following points:

1. The regions within an image are not uniform; some regions are smooth while some are complex. Block based histogram shifting schemes did not consider the complexity of the blocks in the embedding process.
2. Usually in the existing histogram based shifting schemes, maxima and minima are selected to shift the in-between pixels to embed the data. The maxima and the minima point are selected corresponding with the pixels having maximum and minimum frequency value respectively in the image. Out of the various minima (zero) points, any random minima is selected in these schemes. If this is selected judiciously, shifting can be minimized, thereby, minimizing image distortion.
3. In the existing difference image schemes, negative maxima points have not been utilized for data embedding, i.e., in the existing histogram shifting approaches only the absolute differences between the neighboring pixels have been used for data embedding.

1.2 Contribution

The novelty of the proposed scheme is summarized as follows:

1. Complexity of the blocks is evaluated by using Bit Plane Complexity Segmentation (*BPCS*). An optimized threshold value is evaluated by using firefly algorithm and is used to classify a block as smooth or complex. The complex blocks have been used to embed the data, which helps in enhancing the embedding capacity as well as the imperceptibility.
2. Maxima and minima (zero) points of difference blocks are chosen in such a way that the distance between them is least, due to which the shifting of the pixels get minimized, hence reduction in distortion, thereby enhancing the visual quality of the image.
3. In the proposed scheme, the difference blocks contain simple difference values, i.e. positive as well as negative difference values, in which the negative difference value pixels are able to store more than one bit of secret data, which enhances the embedding capacity accordingly maintaining the similar visual quality of the image.

1.3 Organization of the Work

The paper has been organized into following sections. Section 2 discusses the background of the proposed scheme. The proposed scheme itself, which comprises of the schemes of data embedding, and data extraction and handling of the side information are discussed in Section 3. Experimental results and comparisons of the proposed scheme with existing schemes have been illustrated in Section 4 followed by the conclusion in Section 5.

2 RELATED WORK

The concepts of histogram shifting, image complexity and firefly algorithm, used in the proposed scheme, are discussed in this section.

2.1 Histogram Shifting

Reversible Data Hiding using Histogram Shifting was proposed by Ni et al. [14] in 2006. The main concept of the shifted-histogram data hiding method is to find a pair of maxima and minima in the histogram of the image and then shift the intensity of the pixels, which lie between the maxima and minima by one level, towards the minima. If the minima lies to the right of the maxima, all the pixel values between them are incremented by 1, while if the minima lies to the left of the maxima, all the values are decremented by 1, as shown in the Figure 1. This creates an empty space on the shifted histogram around the maxima pixel value. To embed a data stream, the shifted image is re-scanned and when the pixel of maximum frequency is encountered its gray value is incremented by one, if the corresponding bit in the embedding stream is '1' otherwise it remains unaltered. Thus, the largest number of bits which can be hidden into the image is equal to the maximum frequency of the histogram. Owing to the created gap, the data hiding mechanism is reversible. The values of the pixels with maximum and minimum frequency are also recorded as side information. If the minimum frequency is non-zero, then their numbers also need to be embedded as the side information, which reduces the data hiding capacity of the system.

2.2 *RDH* Based on Histogram Modification of Difference Images

In this section, Lin et al. scheme [11] is briefly reviewed in which the cover image is divided into non-overlapping blocks of equal size and difference image is generated for every block. In the histogram based data hiding schemes, the higher the number of peak points, the larger will be its hiding capacity. With the help of the histogram approach, peak values are recorded for every block from its difference image. In the difference image, the grayscale value of the maximum occurrence tends to be around 0. The objective in their scheme is to increase the occurrence of peak point

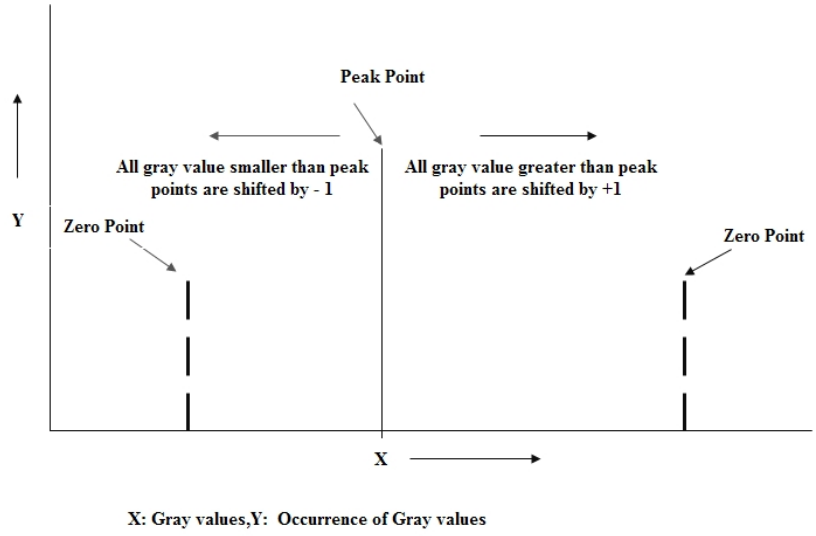


Figure 1. Shifting of pixels in the histogram of an image

and hence the hiding capacity of the proposed scheme. During the hiding phase, empty space is created in the difference image by shifting the histogram and hiding the secret data by using the histogram shifting process. The marked image is generated using original cover image and modified difference image. For the receiver part, same steps are applied in the reverse order to retrieve the original image from the marked image.

2.3 Image Complexity

The data hiding scheme outlined in this paper uses the complex region of the image to embed data. Image complexity has no standard measure. Kawaguchi [9] proposed the measure to evaluate image complexity, which is also known as black-and-white (B-W) border image complexity.

The black-and-white border length is recorded in the binary image to measure the image complexity. The longer border indicates complex image, else it is a simple image. The length of the border is calculated by the summation over the number of times the color changes along all the rows and columns of the image. For example, if a black pixel is surrounded by white background pixels, then it has the border length of 4. The image complexity is defined as follows:

$$\alpha = \frac{k}{m} \tag{1}$$

where m is maximum possible B-W changes in the image and k is the total length

of B-W border in the image. Hence the value ranges over, $0 \leq \alpha \leq 1$. The above complexity measure is defined only for binary images. The 8-bit grayscale images are split into 8 binary planes. This splitting of image into its constituent binary planes is called Bit-Plane Slicing. This is performed in Pure-Binary Coding system (PBC) where the intensity values are represented as 8 bit binary numbers, but it suffers from a serious drawback, known as Hamming Cliff [20]. This issue is resolved by using the coding system called Canonical Gray Coding System (CGC), where successive decimal numbers differ in their representation by just one bit. Thus in $BPCS$ firstly the absolute intensity values are converted into CGC by PBC-to-CGC mapping. This is followed by bit-plane decomposition on the CGC values, and the 8 binary images obtained are called the CGC images. These CGC images do not suffer from Hamming cliffs.

The complexity measure for the grayscale image is explained with the help of an example in Figure 2. An image block and its binary representation is shown in the figure. This block is divided into 8 bit planes. k_i is calculated for every block, where $i \in [1, 8]$. k_i is length of B-W border which equals summation of number of color changes along the rows and columns in the image block. Maximum length (m) = size of image $\times$ number of bit planes. For the given image block, size of image = 3×3 and number of bit planes = 8. Then, complexity (α) is calculated using Equation (1).

2.4 Firefly Algorithm

Firefly Algorithm (FA), proposed by Yang et al. [25], is a swarm intelligence optimization technique. FA is inspired by flashing behavior of fireflies. Two basic functions of the flash light are to attract mating partners and to attract potential prey. FA is based upon the assumption that solution of an optimization problem can be perceived as fireflies whose brightness is proportional to the value of its objective function within a given problem space. In the FA, there are three idealized rules:

1. All fireflies are unisexual, so that any individual firefly will be attracted to all other fireflies.
2. Attractiveness is proportional to their brightness, and for any two fireflies, the less bright one will be attracted by (and thus move towards) the brighter one; however, the intensity (apparent brightness) decreases as their mutual distance increases.
3. If there are no fireflies brighter than a given firefly, it will move randomly.

3 PROPOSED SCHEME

In this section, the data embedding algorithm, data extraction algorithm and prevention of overflow or underflow has been discussed.

167	133	111
144	140	135
159	154	148

Image Block

10100111	10000101	01101111
10010000	10001100	10000111
10011111	10011010	10010100

Binary Representation of Image Block

1	1	0
1	1	1
1	1	1

$k_1 = 2$

0	0	1
0	0	0
0	0	0

$k_2 = 2$

1	0	1
0	0	0
0	0	0

$k_3 = 4$

0	0	0
1	0	0
1	1	1

$k_4 = 4$

0	0	1
0	1	0
1	1	0

$k_5 = 7$

1	1	1
0	1	1
1	0	1

$k_6 = 6$

1	0	1
0	0	1
1	1	0

$k_7 = 8$

1	1	1
0	0	1
1	0	0

$k_8 = 6$

Bit Planes and their corresponding B-W changes

Calculation for complexity of a block:

$m = 72$

$k = k_1 + k_2 + ... + k_8$

$= 2 + 2 + 4 + 4 + 7 + 6 + 8 + 6$

$= 39$

complexity = k / m

= 39 / 72

= 0.541

Figure 2. Representation of bit plane slicing and complexity measure

Algorithm 1 Firefly algorithm for maximum optimization

```
Objective function: f(x), where x = (x_1, x_2, x_3, ..., x_t);
Generate Initial Population of fireflies x_i where i = 1, 2, 3, ..., t;
Formulate light intensity I in association with f(x)
Define absorption coefficient γ
while (t < MaxGeneration) do
    for i ← 1 to t do
        for j ← 1 to t do
            if (I_i > I_j) then
                Vary attractiveness with distance r via exp(−γr);
                move firefly i towards j
                Evaluate new solutions and update light intensity
    Rank fireflies and find current best
```

3.1 Data Embedding

Let CI be the cover image of size $M \times N$ and SI be the Secret Data. A functional block diagram of the embedding procedure is represented in Figure 3 and discussed in Algorithm 2.

Algorithm 2 Embedding Algorithm

Input: Cover Image CI of size $M \times N$, Secret Data Image SI
Output: Marked Image MI

1: The cover image CI is divided into blocks CI_b of size $r \times c$. The number of blocks n created for the image would be: $n = \dfrac{M \times N}{r \times c}$
2: Complexity value is calculated for all the blocks using Kawaguchi et al. scheme [9], described in Section 2.3. Then blocks are categorized into two categories: Complex and Smooth. The set of blocks having complexity value greater than the threshold are considered as Complex Blocks while the set of blocks having complexity value lesser than the threshold are considered as Smooth Blocks. An optimized threshold value is evaluated by using *FA*, mentioned in Algorithm 1. The set of complex blocks is used for data embedding.

An example representing embedding algorithm has been shown in Figure 4.

3.2 Data Extraction

In this procedure, the embedded secret data bits are extracted from the marked image MI and it is restored to its original image without any distortion. The steps within the extraction procedure are discussed in Algorithm 3:

3: From every selected complex block CI_b, absolute difference image block CAD_b of size $r \times (c-1)$ is evaluated by using the formula:

$$CAD_b(i,j) = |CI_b(i,j) - CI_b(i,j+1)|, \tag{2}$$

for $0 \le i \le r-1$, $0 \le j \le c-2$, $1 \le b \le n$

4: A simple difference image CSD_b of size $r \times (c-1)$ of each block b is evaluated by using the formula:

$$CSD_b(i,j) = CI_b(i,j) - CI_b(i,j+1), \tag{3}$$

for $0 \le i \le r-1$, $0 \le j \le c-2$, $1 \le b \le n$

5: For all the absolute difference image blocks CAD_b, histograms are generated and maxima $\max_b$ and minima $\min_b$ are recorded for every block. The minima is chosen in such a way that the value of $|\max_b - \min_b|$ should be minimum. All the maximas are copied from CSD_b to CAD_b so that CAD_b should consist of positive and negative maximas and other pixels as absolute values.

6: All the pixel values between $\max_b$ and $\min_b$ are incremented or decremented by 1 depending on the case that the minima $\min_b$ is present on the right or left of the corresponding maxima $\max_b$. Rest all the pixel values $CAD_b(i,j)$ remain unchanged. Both cases can be mathematically described in Equations (4) and (5) as follows:

$$CAD'_b(i,j) = \begin{cases} CAD_b(i,j) + 1, & \text{if } CAD_b(i,j) > \max_b, \\ CAD_b(i,j), & \text{otherwise,} \end{cases} \tag{4}$$

$$CAD'_b(i,j) = \begin{cases} CAD_b(i,j) - 1, & \text{if } CAD_b(i,j) < \max_b, \\ CAD_b(i,j), & \text{otherwise,} \end{cases} \tag{5}$$

for $0 \le i \le r-1$, $0 \le j \le c-2$, $1 \le b \le n$.

7: The secret data bits m of the secret data, SI, are embedded in the shifted difference image blocks CAD'_b by modifying their pixels having grayscale equal to the maxima $\max_b$, with the help of the following principle:

$$CAD''_b(i,j) = \begin{cases} CAD'_b(i,j) + m, & \text{if } CAD'_b(i,j) = \max_b, \\ |CAD'_b(i,j)| + m_1, & \text{if } CAD'_b(i,j) = -\max_b, \\ CAD'_b(i,j), & \text{otherwise,} \end{cases} \tag{6}$$

for $0 \le i \le r-1$, $0 \le j \le c-2$, $1 \le b \le n$, $m \in \{0,1\}$ and $m_1 \in \{0, 2^L\}$ where $L = \log_2(d)$, d is the difference between negative and positive maximas of the block.

8: The blocks of marked image MI are created with the help of blocks of the original cover image CI and the blocks of the updated difference image CAD'' with the help of the following transformations:

$$MI_b(i,0) = \begin{cases} CI_b(i,0), & \text{if } CI_b(i,0) < CI_b(i,1), \\ CI_b(i,1) + CAD''_b(i,0), & \text{otherwise.} \end{cases} \tag{7}$$

$$MI_b(i,1) = \begin{cases} CI_b(i,0) + CAD''_b(i,0), & \text{if } CI_b(i,0) > CI_b(i,1), \\ CI_b(i,1), & \text{otherwise,} \end{cases} \tag{8}$$

for $0 \le i \le r-1$, $0 \le j \le c-2$, $1 \le b \le n$.

9: For other remaining pixels, the following operation is performed

$$MI_b(i,j) = \begin{cases} MI_b(i,j-1) + CAD''_b(i,j-1), & \text{if } CI_b(i,j-1) \ge CI'_b(i,j), \\ MI_b(i,j-1) - CAD''_b(i,j-1), & \text{otherwise,} \end{cases} \tag{9}$$

for $0 \le i \le r-1$, $0 \le j \le c-2$, $1 \le b \le n$.

3.3 Preventing Possible Overflow or Underflow

In the marked image, the grayscale values of some pixels of the cover may exceed the upper bound ($2^{bd}-1$ for cover image having bd depth pixels) or the lower bound (0 for cover image having bd depth pixels). This is possible due to the shifting operations performed on pixel values that are close to $2^{bd}-1$ or 0. To overcome the overflow or underflow problem, the modulo operation proposed by Goljan et al. [4] and Honsinger et al. [5], is adopted. For the marked image, we define each pixel $S_b(i,j)$ as

$$S_b(i,j) = S_b(i,j) \bmod 256.$$

On the receiver side, whether the received pixel, for example, $S_b(i,j) = 255$, was derived from 255 or -1, must be distinguished. Considering the characteristics of an image, no tremendous variations exist for adjacent pixels. Therefore, in case of a significant difference between $S_b(i,j-1)$ and $S_b(i,j)$, $S_b(i,j)$ was conducted by a modulo operation.

Two evaluations are presented here to restore the original value of $S_b(i,j)$ after the modulo operation is performed. If $S_b(i,j-1)$ is larger than TH_1, $S_b(i,j)$ is restored as

$$S_b(i,1) = \begin{cases} S_b(i,1) + 256, & \text{if } |S_b(i,j-1) - S_b(i,j)| \ge TH_2, \\ S_b(i,1), & \text{otherwise,} \end{cases}$$

where TH_1 and TH_2 are the threshold values.

Algorithm 3 Extraction Algorithm

Input: Marked Image *MI*

Output: Original image *CI* and secret data image *SI*.

1: The received marked image is divided into n blocks of size $r \times c$.

2: The complexity values are calculated for all the blocks using scheme described in Section 2.3. These blocks are classified into complex and smooth on the basis of their comparison with the threshold value. An optimized threshold value is evaluated using *FA* described in Algorithm 1. The set of blocks having complexity value greater than the threshold are considered as Complex Blocks while the set of blocks having complexity value lesser than the threshold are considered as Smooth Blocks. Only the complex blocks are used for extraction as the data was embedded only in the complex blocks.

3: Difference image complex blocks $RCAD_b$ are calculated from the received marked image by using the formula:

$$RCAD_b(i,j) = |MI_b(i,j) - MI_b(i,j+1)| \tag{10}$$

for $0 \leq i \leq r-1$, $0 \leq j \leq c-2$, $1 \leq b \leq n$.

4: Values of maxima $\max_b$ and minima $\min_b$ are recieved as side information.

5: The difference image generated in Step 3 is traversed and the embedded secret data bits m are extracted by using the following rules.

$$m = \begin{cases} 0, & \text{if } RCAD_b(i,j) = \max_b, \\ 1, & \text{if } RCAD_b(i,j) = \max_b + 1, \\ de2bi(RCAD_b(i,j)), & \text{if } RCAD_b(i,j) \in [-\max_b, \max_{b-1}), \end{cases} \tag{11}$$

for $0 \leq i \leq r-1$, $0 \leq j \leq c-2$, $1 \leq b \leq n$ where *de2bi* is the conversion function of any decimal number to its equivalent binary number. $\max_b$ is the received maxima of block b and m is the array of bits extracted. The entire difference image blocks b are scanned and bit 0 is extracted, if the pixel value $\max_b$ is encountered and 1 is extracted if the pixel value $\max_b + 1$ is encountered. If pixels within the range $[-\max_b, \max_{b+1})$ are encountered, L number of bits are retrieved by following Equation (11), where

$$L = \lfloor \log_2(d) \rfloor \tag{12}$$

L is the number of bits embedded in every negative maxima and

$$d = \max_b - (-\max_b) + 2 \tag{13}$$

$\max_b$ is the positive peak value and $-\max_b$ is the negative peak value.

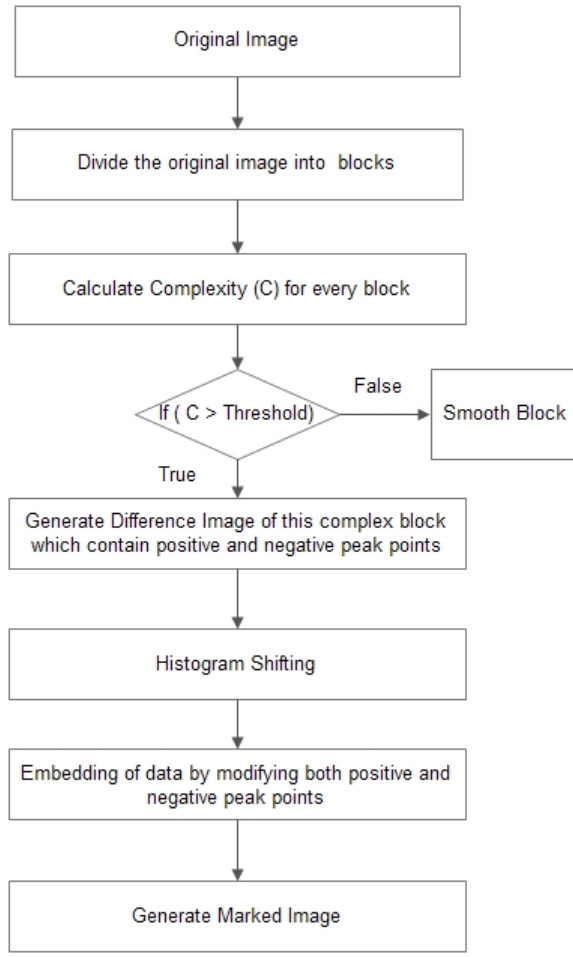


Figure 3. Flowchart of data embedding procedure

6: The difference image blocks $RCAD'_b$ are retrieved back by using the formula:

$$RCAD'_b = \begin{cases} RCAD_b(i,j) - 1, & \text{if } RCAD_b(i,j) = \max_b + 1, \\ RCAD_b(i,j), & \text{if } RCAD_b(i,j) = \max_b, \\ -\max_b, & \text{if } RCAD_b(i,j) \in [-\max_b, \max_{b-1}), \end{cases} \tag{14}$$

for $0 \leq i \leq r-1$, $0 \leq j \leq c-2$, $1 \leq b \leq n$ where $\max_b$ is the maxima of block b and $-\max_b$ is the negative maxima of block b.

7: The original image CI is recovered back by applying the inverse shifting operations.

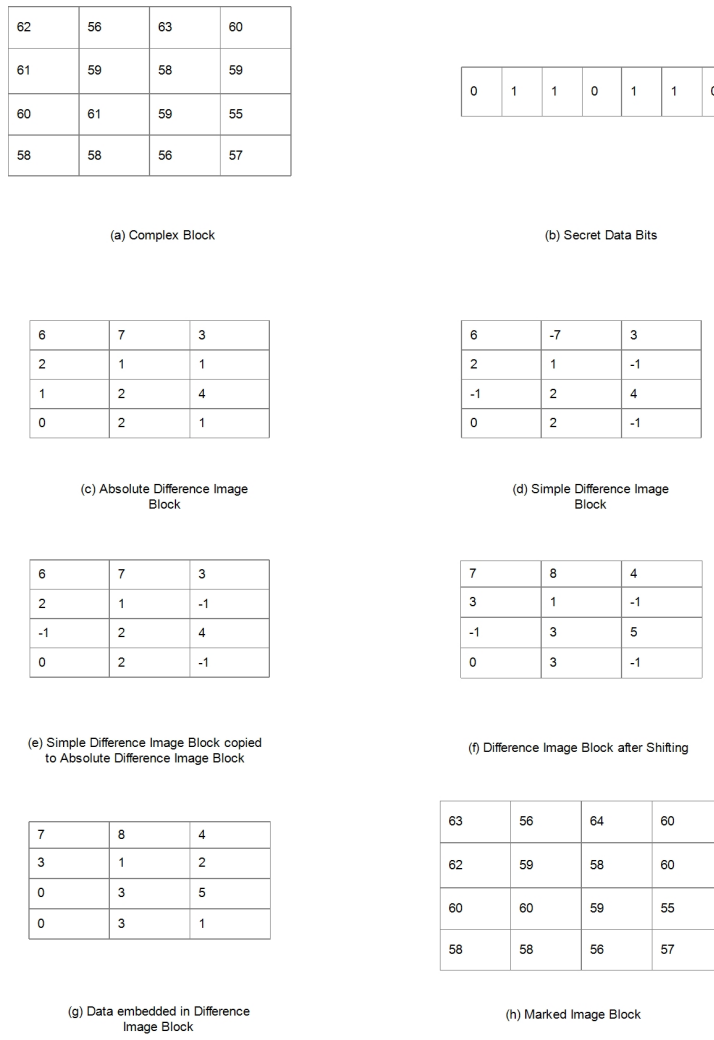


Figure 4. Example for data embedding in proposed scheme

If $S_b(i, j-1)$ is smaller than TH_1, $S_b(i, j)$ is restored as

$$S_b(i,1) = \begin{cases} S_b(i,1) - 256, & \text{if } |S_b(i,j-1) - S_b(i,j)| \geq TH_2, \\ S_b(i,1), & \text{otherwise.} \end{cases}$$

This approach is an efficient approach to prevent underflow and overflow and is also used by Lin et al. [11].

4 EXPERIMENTAL RESULTS

Figure 5. a)–e) Original cover images; f)–j) marked images after embedding 5 000 bits; k)–o) marked images after embedding 10 000 bits; p)–t) marked images after embedding 25 000 bits

Image	Capacity → / Threshold ↓	5 000	10 000	20 000	50 000
Lena	0.0983	64.65	61.43	58.61	54.86
Boat	0.0635	65.68	62.65	59.83	56.10
Baboon	0.065	67.42	63.84	60.67	55.94
Barbara	0.0762	48.40	45.26	42.11	37.81
Peppers	0.0683	64.59	61.46	58.40	54.57

Table 1. *PSNR* (dB) values at different capacities (in bits) for different images at optimized threshold

Proposed scheme is implemented using *MATLAB*. The input images considered for this work are uncompressed grayscale images named Lena, Boat, Peppers, Baboon and Barbara. These images are generally used by the researchers to compare their results with existing schemes. Each of the image is 512×512 size and they are shown in Figure 5. Figures 5 a)–e) show the original images and their marked images at different embedding data, are shown in Figures 5 f)–t). Block sizes considered in

Image →	Lena		Barbara		Boat		Baboon		Peppers	
Scheme ↓	Capacity	*PSNR*	Capacity	*PSNR*	Capacity	*PSNR*	Capacity	*PSNR*	Capacity	*PSNR*
Tsai et al. [23]	38 310	49.11			25 370	48.97	12 739	48.85		
Kim et al. [10]	78 071	41.09			52 924	40.65			65 293	40.84
Hong et al. [6]	54 457	48.17			34 148	48.17			34 025	48.77
Ni et al. [14]	5 446	48.21			11 473	48.26			5 447	48.21
Ou et al. [16]	60 000	50.5	47 500	51.1	–	–	20 000	50.1		
Ma el al. [13]	50 000	50.5	40 000	51.5			17 000	50	38 000	50.5
Qu and Kim el al. [19]	49 000	50.5	38 000	50.5			15 000	50.5	34 000	51.5
Ou et al. [17]	54 000	49	43 000	49			18 000	49	43 000	49
Hong et al. [8]	46 839	49.19			29 824	49.02	14 154	48.86		
Proposed	102 431	51.85	154 237	33.01	91 687	53.23	88 807	53.28	97 767	51.60

Table 2. *PSNR* (dB) vs. Capacity (in bits) comparison of proposed scheme with the existing schemes

Block Size →	4×4		8×8		16×16	
Image ↓	Capacity	*PSNR*	Capacity	*PSNR*	Capacity	*PSNR*
Lena	102 431	51.85	77 206	47.97	65 496	43.30
Baboon	88 807	53.28	54 817	50.54	40 932	46.58
Peppers	97 767	51.60	72 332	47.89	61 611	43.35
Barbara	154 237	33.01	134 238	27.57	113 465	23.18
Boat	91 687	53.23	62 294	49.96	40 086	45.77

Table 3. *PSNR* (dB) vs. Capacity (in bits) comparison of proposed scheme for different block sizes

the proposed work are 4×4, 8×8 and 16×16. In this experiment, a binary image is taken as secret data. *PSNR* is taken as one of the quality parameters which is defined as

$$PSNR = 10 \times \log_{10} \frac{(2^{bd} - 1)^2}{MSE} \tag{15}$$

where bd is the bit depth of the image and MSE is defined as

$$MSE = \sum_{i=1}^{M} \sum_{j=1}^{N} \frac{\delta(i,j)^2}{M \times N}, \tag{16}$$

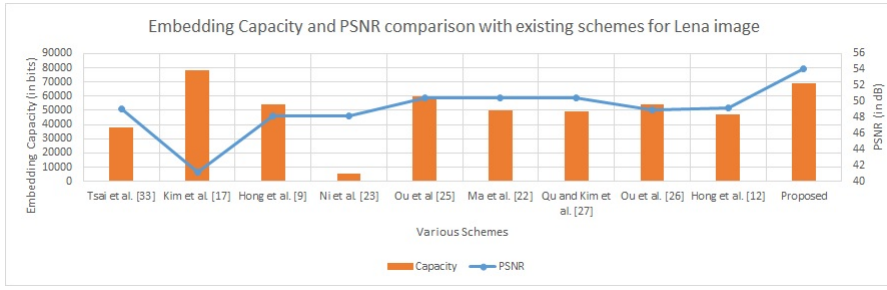
$$\delta(i,j) = MI(i,j) - CI(i,j) \tag{17}$$

Schemes	Sachnev et al. [21]	Ma et al. [13]	Qu et al. [19]	Ou et al. [16]	Proposed Scheme 4×4	Proposed Scheme 8×8	Proposed Scheme 16×16
Lena	20 785	144 390	10 370	10 290	4 758	3 027	1 440
Baboon	60 312	53 715	57 549	52 008	4 266	4 257	7 821
Barbara	21 068	16 177	11 512	10 223	2 093	798	420
Peppers	42 296	23 239	17 741	16 369	5 048	4 347	4 895

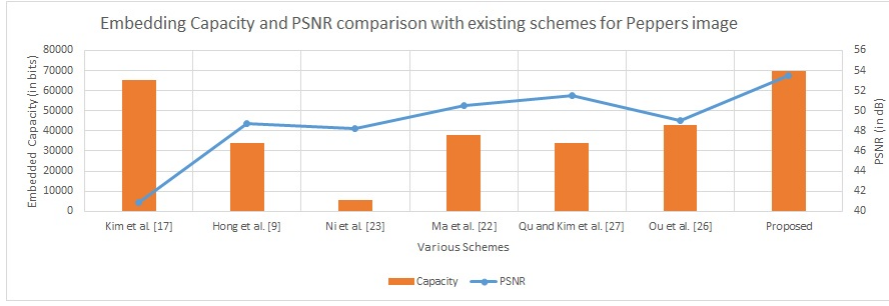
Table 4. Shifting (in pixels) comparison for different images for 10 000 embedding capacity

Schemes	Sachnev et al. [21]	Ma et al. [13]	Qu et al. [19]	Ou et al. [16]	Proposed Scheme 4×4	Proposed Scheme 8×8	Proposed Scheme 16×16
Lena	43 420	33 795	27 176	26 825	9 879	7 085	3 540
Barbara	43 420	34 371	29 644	28 420	4 027	1 636	909
Peppers	90 388	52 460	43 250	43 191	10 012	8 675	5 089

Table 5. Shifting (in pixels) comparison for different images for 20 000 embedding capacity



a)



b)

Figure 6. Embedding capacity (in bits) and *PSNR* (dB) comparison of proposed scheme with existing schemes for a) Lena image and b) Peppers image

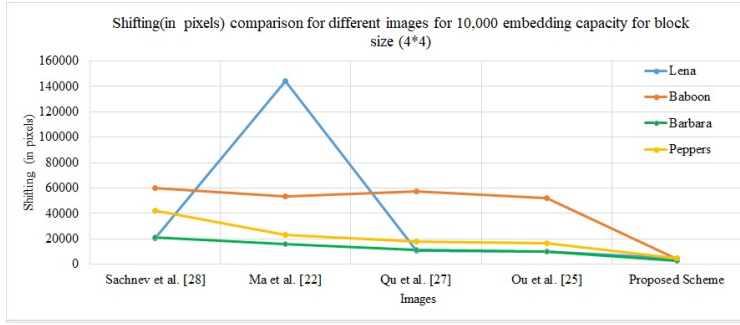
where $MI(i,j)$ is the pixel of marked image and $CI(i,j)$ is the pixel of cover image, M and N are the height and width of image, respectively.

Objective function used to optimize the threshold is

$$f_{obj} = \text{embedding\_capacity} \times PSNR \tag{18}$$

where embedding_capacity is the total amount of secret data embedded and *PSNR* is taken between original and marked image.

In Table 1, *PSNR* values have been given for the proposed scheme for different embedding capacities for different images at their respective optimized thresholds.



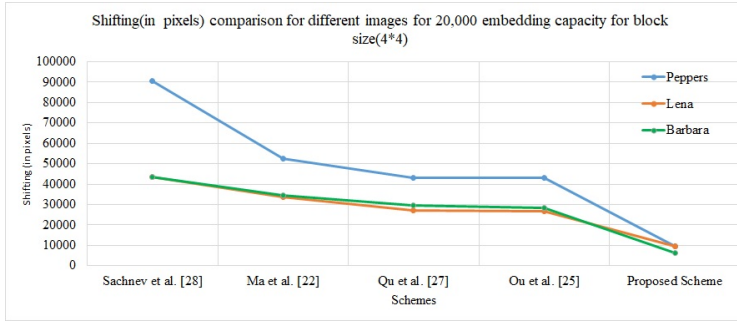
a)

Minimum bits embedded are 5 000 and maximum bits embedded are 50 000 for all images, considered in this work. One can observe that *PSNR* values decrease with the increase in the embedded bits. This is due to the reason as number of embedded bits decreases the distortion in the marked image, which further decreases *PSNR* between cover and marked images.

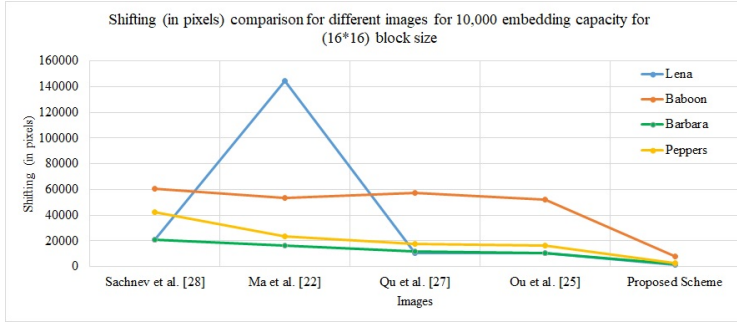
In Table 2, the *PSNR* vs. the maximum embedding capacity of the proposed scheme and various existing schemes have been presented. It can be inferred that the embedding capacity and *PSNR* of the proposed scheme are better when compared to the existing schemes. The embedding capacity has been increased using negative pixels of the difference image for embedding. *PSNR* has been maintained by choosing the closest minima to the maxima of every difference block of the image and also by using just the complex blocks for the host image. In case of Barbara image, *PSNR* value is lesser than the existing schemes, but the embedding capacity has been increased remarkably as compared to the existing schemes. Table 3 shows *PSNR* and embedding capacity for the proposed scheme for different block sizes, i.e. 4×4, 8×8 and 16×16. In Tables 4 and 5, the comparison of the shifting of pixels of the proposed scheme with the existing schemes at 10 000 and 20 000 embedding capacities has been shown. It can be seen that the shifting of pixels in the proposed scheme has been reduced remarkably in comparison to the shifting of the existing schemes.

PSNR vs. embedding capacity comparison of the proposed scheme with the existing schemes has been shown in Figure 6 for Lena and Pepper images. It can be observed from this figure that the proposed scheme maintains the highest *PSNR* vs. embedding capacity ratio for both the images. Though Kim et al. [10] manage to have larger embedding capacity for Lena image, but its *PSNR* for the same is quite low as compared to the proposed scheme.

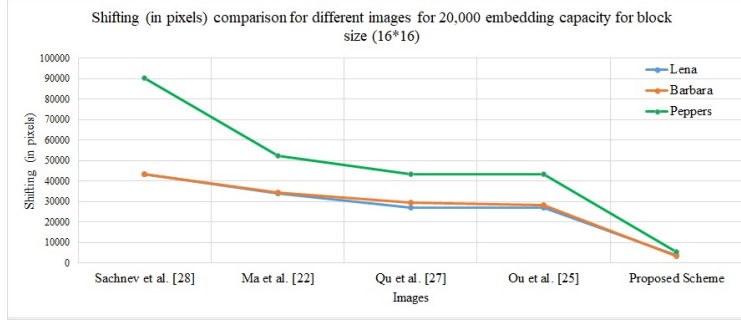
Comparison of the proposed scheme with the existing schemes in terms of shifting of pixels has been shown in Figure 7. It can be observed that the proposed scheme has minimum shifting of pixels, which helps in reducing the distortion, thereby enhancing the visual quality of the image.



b)



c)



d)

Figure 7. Shifting (in pixels) comparison of the proposed scheme with the existing schemes for different images for a) block size 4×4 and embedding capacity 10 000, b) block size 4×4 and embedding capacity 20 000, c) block size 16×16 and embedding capacity 10 000, and d) block size 16×16 and embedding capacity 20 000

5 CONCLUSION

In this paper, an adaptive reversible data hiding scheme for digital images has been proposed. The blocks of the image have been categorized into complex and smooth blocks on the basis of their complexity, and only complex blocks have been used for embedding, which helped in maintaining the *PSNR* value. Secret data bits are embedded in the difference blocks of the image, in which both the positive and negative difference maxima have been utilized for data embedding, as they are able to embed more than one bit, thereby increasing the embedding capacity. Histogram shifting has been minimized by choosing the closest pairs of maxima and minima which further minimizes the distortion in the marked image. The embedding capacity and the *PSNR* of the proposed scheme have proven to be better as compared to the existing schemes.

REFERENCES

[1] Celik, M. U.—Sharma, G.—Tekalp, A. M.—Saber, E.: Reversible Data Hiding. Proceedings IEEE International Conference on Image Processing, Rochester, NY, 2002, pp. 157–160.

[2] Chang, Y.-F.—Tai, W.-L.: Histogram Based Reversible Data Hiding Based on Pixel Differences with Prediction and Sorting. KSII Transactions on Internet and Information Systems, Vol. 6, 2012, No. 2, pp. 3100–3116, doi: 10.3837/tiis.2012.12.004.

[3] Fallahpour, M.—Sedaaghi, M. H.: High Capacity Lossless Data Hiding Based on Histogram Modification. IEICE Electronics Express, Vol. 4, 2007, No. 7, pp. 205–210, doi: 10.1587/elex.4.205.

[4] Goljan, M.—Fridrich, J. J.—Du, R.: Distortion-Free Data Embedding for Images. In: Moskowitz, I. S. (Ed.): Information Hiding (IH 2001). Springer, Berlin, Heidelberg, Lecture Notes in Computer Science, Vol. 2137, 2001, pp. 27–41, doi: 10.1007/3-540-45496-9_3.

[5] Honsinger, C. W.—Jones, P. W.—Rabbani, M.—Stoffel, J. C.: Lossless Recovery of an Original Image Containing Embedded Data. United States Patent No. US6278791B1, August 2001.

[6] Hong, W.—Chen, T.-S.—Shiu, C.-W.: Reversible Data Hiding for High Quality Images Using Modification of Prediction Errors. Journal of Systems and Software, Vol. 82, 2009, No. 11, pp. 1833–1842, doi: 10.1016/j.jss.2009.05.051.

[7] Hong, W.—Chen, T.-S.: A Local Variance-Controlled Reversible Data Hiding Method Using Prediction and Histogram-Shifting. Journal of Systems and Software, Vol. 83, 2010, No. 12, pp. 2653–2663, doi: 10.1016/j.jss.2010.08.047.

[8] Hong, W.—Chen, T.-S.—Wu, M.-C.: An Improved Human Visual System Based Reversible Data Hiding Method Using Histogram Modification. Optics Communications, Vol. 291, 2013, pp. 87–97, doi: 10.1016/j.optcom.2012.10.081.

[9] Kawaguchi, E.—Taniguchi, R.-I.: Complexity of Binary Pictures and Image Thresholding – An Application of DF-Expression to the Thresholding Problem. Pro-

ceedings of 8th International Conference on Pattern Recognition (ICPR), Vol. 2, 1986, pp. 1221–1225.

[10] Kim, K.-S.—Lee, M.-J.—Lee, H.-Y.—Lee, H.-K.: Reversible Data Hiding Exploiting Spatial Correlation Between Sub-Sampled Images. Pattern Recognition, Vol. 42, 2009, No. 11, pp. 3083–3096, doi: 10.1016/j.patcog.2009.04.004.

[11] Lin, C.-C.—Tai, W.-L.—Chang, C.-C.: Multilevel Reversible Data Hiding Based on Histogram Modification of Difference Images. Pattern Recognition, Vol. 41, 2008, No. 12, pp. 3582–3591, doi: 10.1016/j.patcog.2008.05.015.

[12] Luo, H.—Yu, F.-X.—Chen, H.—Huang, Z.-L.—Li, H.—Wang, P.-H.: Reversible Data Hiding Based on Block Median Preservation. Information Sciences, Vol. 181, 2011, No. 2, pp. 308–328, doi: 10.1016/j.ins.2010.09.022.

[13] Ma, X.—Pan, Z.—Hu, S.—Wang, L.: High-Fidelity Reversible Data Hiding Technique Based on Multi-Predictor Sorting and Selecting Mechanism. Journal of Visual Communication and Image Representation, Vol. 28, 2015, pp. 71–82, doi: 10.1016/j.jvcir.2015.01.012.

[14] Ni, Z.—Shi, Y. Q.—Ansari, N.—Su, W.: Reversible Data Hiding. IEEE Transactions on Circuits and Systems for Video Technology, Vol. 16, 2006, No. 3, pp. 354–362, doi: 10.1109/TCSVT.2006.869964.

[15] Ni, Z.—Shi, Y. Q.—Ansari, N.—Su, W.—Sun, Q.—Lin, X.: Robust Lossless Image Data Hiding Designed for Semi-Fragile Image Authentication. IEEE Transactions on Circuits and Systems for Video Technology, Vol. 18, 2008, No. 4, pp. 497–509, doi: 10.1109/TCSVT.2008.918761.

[16] Ou, B.—Li, X.—Wang, J.: Improved PVO-Based Reversible Data Hiding: A New Implementation Based on Multiple Histograms Modification. Journal of Visual Communication and Image Representation, Vol. 38, 2016, pp. 328–339, doi: 10.1016/j.jvcir.2016.03.011.

[17] Ou, B.—Li, X.—Wang, J.—Peng, F.: High-Fidelity Reversible Data Hiding Based on Geodesic Path and Pairwise Prediction-Error Expansion. Neurocomputing, Vol. 226, 2017, pp. 23–34, doi: 10.1016/j.neucom.2016.11.017.

[18] Pan, Z. B.—Hu, S.—MA, X. X.—Wang, L. F.: Reversible Data Hiding Based on Local Histogram Shifting with Multilayer Embedding. Journal of Visual Communication and Image Representation, Vol. 31, 2015, pp. 64–74, doi: 10.1016/j.jvcir.2015.05.005.

[19] Qu, X.—Kim, H. J.: Pixel-Based Pixel Value Ordering Predictor for High-Fidelity Reversible Data Hiding. Signal Processing, Vol. 111, 2015, pp. 249–260, doi: 10.1016/j.sigpro.2015.01.002.

[20] Richard, E.: A Tutorial on BPCS Steganography and Its Applications. Proceedings of Pacific Rim Workshop on Digital Steganography, Kitakyushu, Japan, 2003, pp. 18–31 (invited paper).

[21] Sachnev, V.—Kim, H. J.—Nam, J.—Suresh, S.—Shi, Y. Q.: Reversible Watermarking Algorithm Using Sorting and Prediction. IEEE Transactions on Circuits and Systems for Video Technology, Vol. 19, 2009, No. 7, pp. 989–999, doi: 10.1109/TCSVT.2009.2020257.

[22] Sencar, H. T.—Ramkumar, M.—Akansu, A. N.: Data Hiding Fundamentals

and Applications: Content Security in Digital Multimedia. Elsevier Academic Press, 2004.

[23] Tsai, P.—Hu, Y.-C.—Yeh, H.-L.: Reversible Image Hiding Scheme Using Predictive Coding and Histogram Shifting. Signal Processing, Vol. 89, 2009, No. 6, pp. 1129–1143, doi: 10.1016/j.sigpro.2008.12.017.

[24] Wang, Z.-H.—Lee, C.-F.—Chang, C.-Y.: Histogram Shifting Imitated Reversible Data Hiding. Journal of Systems and Software, Vol. 86, 2013, No. 2, pp. 315–323, doi: 10.1016/j.jss.2012.08.029.

[25] Yang, X.-S.: Nature-Inspired Metaheuristic Algorithms. Luniver Press, 2008. ISBN: 978-1-905986-10-6.

Sonal Kukreja graduated and received her M.Tech. degree in 2015 in the field of computer science and applications from the Thapar Institute of Engineering and Technology, Patiala. In 2016, she enrolled as doctoral student at the Thapar Institute of Engineering and Technology, Patiala, Punjab, India. Her research interests are focused on data hiding, visual cryptography and watermarking.

Geeta Kasana is working as Assistant Professor in Computer Science and Engineering Department, Thapar Institute of Engineering and Technology, Patiala, India. She received her Ph.D. degree in information security from the Thapar University. Her research interests include image processing and information security. She has published many research papers in reputed international journals and conferences. She is currently guiding Ph.D. students on information security.

Singara Singh Kasana is working as Associate Professor in Computer Science and Engineering Department, Thapar Institute of Engineering and Technology, Patiala, India. He has fifteen years of teaching and research experience. He received his Ph.D. degree in image compression from the Thapar University. His research interests include image processing, wireless networks, and information security. He has published many research papers in reputed international journals and conferences. He is currently guiding Ph.D. students on information security, image and video watermarking, cryptography and remote sensing.

Computing and Informatics, Vol. 38, 2019, 343–366, doi: 10.31577/cai_2019_2_343

MULTI-CARRIER STEGANOGRAPHIC ALGORITHM USING FILE FRAGMENTATION OF FAT FS

Liberios Vokorokos, Branislav Madoš, Norbert Ádám
Anton Baláž, Jaroslav Porubän, Eva Chovancová

Department of Computers and Informatics
Faculty of Electrical Engineering and Informatics
Technical University of Košice
Letná 9, 042 00 Košice, Slovak Republic
e-mail: {liberios.vokorokos, branislav.mados, norbert.adam,
anton.balaz, jaroslav.poruban, eva.chovancova}@tuke.sk

Abstract. Steganography is considered to be not only a science, but also a craft of concealing ongoing communication by hiding messages in unsuspicious cover documents, such as texts, digital images, audio and video sequences. Its essential feature is the constant search for – often exceptionally creative – possibilities of concealing information. In computers, steganography often uses secondary memory and exchangeable memory media utilising file systems. This paper deals with the current state of the issues related to information hiding by means of hard disks, being the most important source of forensic data. This paper focuses on information hiding using the File Allocation Table (FAT) file system. It also proposes a novel multi-carrier algorithm of hiding information in file fragmentation. The algorithm provides flexibility of encoding the information to be hidden and makes steps toward optimization that allows reduction of interference with the current state of the file system, represented by the statistical values of the file fragmentation parameters.

Keywords: Steganography, file system, file allocation table, FAT, fragmentation

Mathematics Subject Classification 2010: 68-R10

1 INTRODUCTION

Currently, computer system security and telecommunications' specialists are strongly focusing on encryption. This method of protection, if used as the sole solution, is considered to be insufficient in certain cases. Encrypted information attracts attention, and sometimes the existence of encrypted communication may even represent a piece of very valuable information. A solution to this problem may be the use of steganography. Steganography can be defined as the art of hiding the presence of communication by embedding secret messages into innocent, innocuos looking cover documents, such as texts, digital images, sound and video files [1].

Steganography has seen a significant development with the advent of computers, computer networks and especially the Internet; this development includes also the introduction of specific steganographic procedures, connected exclusively to the use of computers. Digital steganography employs traditional digital media, such as text, bitmap or vector graphic images, audio and video files. Excellent carriers of concealed messages are graphic image files, most frequently using the following techniques: Least Significant Bit (LSB) modification, frequency domain techniques [2, 3] and spread spectrum techniques. An overview of these techniques may be found in [4]. However, steganography is constantly seeking new, yet unused media and communication channel types, providing a possibility to expand to further territories. An example of this is the appearance of mobile phones, especially smartphones equipped with modern operating systems, such as Android or iOS. An overview on the use of steganography in smartphones is available in [5]. Very useful general overview of the development of steganography is available in [6, 7].

This paper focuses on the use of secondary storage devices and especially file systems to hide confidential information into file fragmentation. It also describes a multi-carrier algorithm which, unlike the other available algorithms, uses a set of files to conceal the information. On the contrary to the existing solutions – such as the steganographic file system proposed in [8], which adds further files to the file system and increases the fragmentation of files – the algorithm presented in this paper aims to minimize the interference with the fragmentation of files and allows to make steps to keep the statistical data concerning file fragmentation untouched, both in the whole file system and in the subset of files used to encode the information. It does not require placing fragments of two or more files into relative positions (i.e. as interlaced files) in the same part of the file system, as the solution proposed by Morkevičius et al. in [9].

In spite of the fact that the FAT file system is not a default file system in modern operating systems anymore, they still support it. Moreover, the FAT file system is still widely used on diskettes, pen drives, various memory media using flash memory chips, as well as solid-state disks (SSD). It is also utilised in numerous types of mobile devices, including mobile phones, MP3 players, cameras, embedded devices and consumer electronics devices, such as set-top boxes and multimedia players. In industry and research and development area software solutions along with specialized hardware are used with secondary storage [10, 11, 12] that often implements FAT

as file system because of relatively easy implementation. Therefore we selected the FAT32 variant of the file system to verify the possibilities of implementation of the algorithm proposed in this paper.

The rest of this paper is organised as follows:

- Section 2 includes the works related to the usage of hard disks and file systems for the purposes of steganography and cryptography, focusing on the algorithms utilising the FAT table and file fragmentation to conceal the information.
- Section 3 contains the proposal of the algorithm developed within this research, allowing hiding information into the file allocation table of the FAT file system. The algorithm encodes the information in the fragmentation of a set of files stored in the file system.
- Section 4 discusses the possibilities of making steps toward optimisation of the encoding of the confidential message with the goal to minimise changes in the file fragment parameters, as well as the possibilities of compensating these changes while keeping the original statistical values of the file fragmentation parameters after the encoding of the confidential message as much as possible.
- Finally, Section 5 lists the achieved results and shows the future research perspectives within this field.

2 RELATED WORKS

Storage devices utilising file systems – especially hard disks – are still the most important storage media used both in enterprise-grade and consumer-grade equipment. Significant amounts of confidential data – both private data and also strategically and financially valuable data of businesses and the state administration – are stored on hard disks [13]. Thus, hard disks belong to the most important sources of forensic analysis. Both the physical and logical structure of hard disks and the methods of storing information on the disk allow relatively many ways of concealing information. These approaches allow hiding the existence of the hidden data from the operating system or the users employing conventional file managers and other software working with folders and files of the file system. Therefore, special software is needed for this purpose. Some of these methods allow fully transparent disk usage, i.e. avoiding random destruction of the hidden information due to conventional usage of the media. On the other hand, other approaches may be vulnerable to random destruction of the concealed data by standard use.

Hard disk drives (HDD) and solid-state drives (SSD) may contain a so-called Host Protected Area (HPA), commonly referred to as "Hidden Protected Area". This part of the disk is not available to the user, to the BIOS, the operating system (OS) or any not-HPA-aware standard software. Therefore, the content of this area may not be read or modified using standard methods. Computer manufacturers may use the HPA to store data protected from the interference of normal users, such as diagnostic software or software used to restore the standard software installation to

its factory-state. It is possible to create software allowing access to the disk sectors of the aforementioned area, storing information in these and reading information from them. Steganography makes a good use of this [14, 15].

A further possibility of using hard disks in steganography is dividing physical storage space used to store data on the disk – i.e. sectors and clusters – into contiguous regions called partitions. By not including a set of sectors in any partition, these sectors may be used to store data, invisible to the standard access methods of the operating system. This unallocated space is called the *disk slack* or the *volume slack*. The related issues are described in detail in [16].

Since the sectors belonging to the partition may be addressed only by whole clusters and the size of the partition may be defined in such a way that when dividing the number of sectors of the partition by the number of sectors in the cluster the remainder is not an integer, there may remain some sectors at the end of the partition not addressable using standard file system methods, thus, this remaining area may be used to store the confidential information. These sectors are then commonly referred to as the *partition slack* [17].

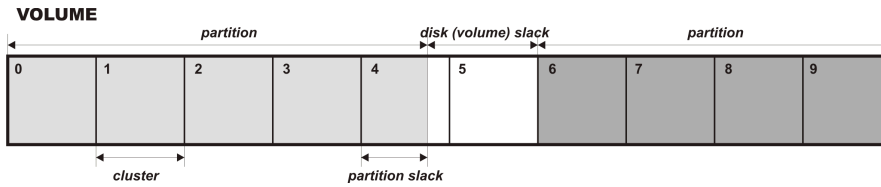


Figure 1. A volume with two partitions, showing the existence of a partition slack and disk slack and/or volume slack

Figure 1 shows a volume having two partitions. The first of them does not occupy the whole last cluster in the volume, so it cannot use it. The allocated but unusable part of the cluster is the *partition slack*. A further part of the disk not allocated to any of the partitions is the *disk slack/volume slack*.

If there are clusters in the partition not used for the storage of regular data – i.e. "empty clusters" – these are not accessible by standard means and thus may be used to store confidential information. Even if this area is large, allowing the storage of huge amounts of data – potentially up to hundreds of GB – any unused cluster may be used anytime to store regular file content and so the hidden data may be overwritten, leading to their loss.

The DOS/Windows operating system reserves the first sector of the hard disk for the Master Boot Record (MBR), which stores the information required to load the operating system and also the disk partitioning information. Even if the size of the MBR itself is small and it takes up only a single sector, the whole track, on which it is stored, is reserved and the sector containing the track cannot be addressed and used by the file system. This allows the existence of an eventually large space on the disk, usually amounting to tens of sectors, which may be used to store the concealed information [18].

Analogous to the MBR is the case of the Extended Master Boot Record (EMBR) of the extended partition. This space is commonly referred to as the *MBR slack* and the *EMBR slack*, respectively. The hidden data, stored in the MBR are protected not only from formatting, but also from the change of partition count and size on the disk.

In the file systems used by the Windows operating system, the size of a cluster may range from 512 B to 64 kB, while each file written into the file system may use one or more clusters. Since the remainder of the division of the length of the stored files in bytes and the size of the cluster is not necessarily 0, the last cluster used to store the file is being used only partly. Therefore, it may happen that one or more sectors in the last cluster are unused and so this space may be used to store the hidden information. This space is called the *file slack*.

Last sector used in the cluster to store the regular data of the file need not be fully used; also this unused space may be used to store hidden information. This space is commonly referred to as the *RAM slack*. The term *RAM slack* is a historical term – in the past, upon writing the last part of the file from the operating memory to the sector on the disk, 512 B of operating memory was copied to this space, even though some bytes had nothing in common with the content of the file. When using the *file slack* or the *RAM slack*, it is very probable that the concealed data shall be overwritten any time the size of the regular file occupying the cluster changes.

Figure 2 shows the FILE.EXT file occupying two clusters (each consisting of four sectors), while the last cluster of the file is not being fully occupied. Its first sector is not completely filled with file data and so the empty part is the *RAM slack*. The following three sectors of the cluster are fully unused – these are the *file slack*.

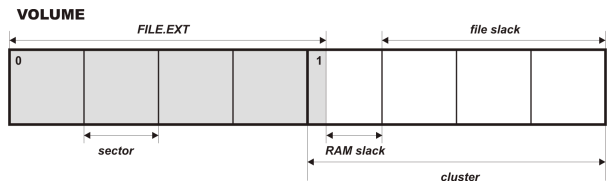


Figure 2. A file occupying two clusters, with the RAM slack and the file slack depicted

In [19], Aycock and de Castro proposed to utilise the fact that the order of files and directories displayed to the user does not correspond to their order of storage in the directories of the File Allocation Table (FAT). Permutations of their storage order allow concealing information.

As a feature of the FAT file system, the FAT table allows marking clusters in the damaged parts of the storage media as bad clusters, indicating their inappropriateness for data storage. By falsely marking fully functional clusters as bad clusters one may ensure that these will not be used by the file system. However, these are

fully functional and one may hide information in these, using a special software tool [19, 20].

A multi-carrier steganographic algorithm, storing information in file fragmentation and the relative location of these fragments in the file system was proposed by Morkevičius et al. in [9].

The utilisation of the file system as a carrier to hide information is being used not only in steganography, but also in cryptography. There are numerous implementations of cryptographic file systems available, such as the Cryptographic File System (CFS) for the UNIX operating system [21], the Transparent Cryptographic File System (TCFS) for the Linux operating system [22], the Encrypting File System (EFS) for the Microsoft Windows 2000/XP operating systems [23] or the Secure File System (SFS) for the same [24, 25]. Further cryptographic file systems include the E4M [26] and PGPDisk [27] systems.

Cryptographic file systems encode individual files and whole disk partitions, protecting user data from unwanted recovery of their content. On the other hand, the use of cryptographic functionality tells the eventual attacker that the data protected are truly confidential and important. Such a situation draws attention of the potential attackers and may inspire them to try to crack the encryption or to force the authorised user to decrypt the data under pressure. A solution to this problem may be the use of steganography, which allows masking the existence of concealed files from unauthorised users. This allows also the use of plausible deniability concept, i.e. allowing the authorised user to deny the existence of confidential data or, eventually, disclose only the existence of less important data. The attacker might not be sure and cannot prove, whether there are any further, more important data, remaining concealed.

Anderson et al. proposed in [28] a steganographic file system. In this, the user may access the requested file only by knowing its name and the appropriate password. The proposed file system inspired Van Schaik and Schmeddle to implement such a steganographic file system for the LINUX operating system [25]. In [29], Hand and Roscoe proposed an enhancement to this scheme for peer-to-peer platforms by replacing simple file replication with the Information Dispersal Algorithm (IDA).

A further implementation of a steganographic file system based on [28] was StegFS, proposed in [30]. This was an extension of the standard file system of the LINUX operating system by encryption functions allowing plausible deniability. An implementation of the steganographic file system for the Windows environment is ScramDisk, presented in [31].

A steganographic file system based on JPEG files was proposed in [32, 33]. It allowed the creation of a virtual disk, a Virtual File System (VFS), hidden in multiple cover media – images in JPEG format. The hidden content is available only to the user knowing the correct key.

The following section describes the proposed algorithm, aimed at hiding information in the FAT file system by using file fragmentation.

3 NEW MULTI-CARRIER FILE FRAGMENTATION BASED STEGANOGRAPHIC ALGORITHM

For the purposes of the algorithm, the information stored in the FAT system (files) form the set V. The total file count is x, thus $x = |V|$ (cardinality). So, the following applies:

$$V = \{F_o, F_1, \ldots, F_{x-1}\}. \tag{1}$$

On hard disks of real-life personal computers (with an installed operating system, software and user data files), the size of this set amounts to hundreds of thousands.

The files stored in the FAT are separated into fragments. A fragment consists of data of a specific file stored in a contiguous sequence of clusters allocated so that in the FAT table the record of the cluster at the address n points to the next cluster at the address $n + 1$. An exception to this rule is the last cluster of the fragment at the address n, pointing to the first cluster of the next fragment, which must not be located at the address $n + 1$. A fragment may be limited from above and from below by unallocated clusters or clusters allocated by another fragment. It may be limited also by other fragments of the same file, while the fragment is not a subset of another fragment. Thus, we may define the set A, formed by all fragments of all files stored in the file system.

$$A = \{f_o, f_1, \ldots, f_{y-1}\}. \tag{2}$$

Obviously, $y = |A|$, while the value depends on the current file fragmentation and is variable. From below, it is limited by the value of x – the number of files stored in the file system – because if a file is not fragmented, we may assume that it consists of a single fragment. If all files were unfragmented, the number of fragments in the file system would be equal to the number of files. In theory, the upper limit of the total fragment count is the sum of clusters allocated to the individual files of the set V, being an extreme state, when each fragment of each file consists of a single cluster. A limiting factor is also the maximum amount of clusters available to the file system. Each fragment may be part only of a single file.

In the proposed algorithm, each $f_z \in A : z \in \langle 0, y - 1\rangle$, $z \in \mathbb{N}^0$ fragment is an ordered set of fragment parameters:

$$f_z = \{\alpha_1, \alpha_2, \alpha_3, \alpha_4, \alpha_5\} \tag{3}$$

where

- α_1 is the address of the first cluster of the fragment in the FAT table,
- α_2 is the length of the fragment in clusters,
- α_3 is the address of the last cluster of the fragment in the FAT table,
- α_4 is the length to the next fragment of the file in clusters,
- α_5 is the address of the first cluster of the next fragment of the file.

In case of the last fragment in the sequence of fragments of the given file, the parameters α_4 and α_5 are set to 0. The set of fragment parameters is not complete, currently it is only a proof of concept.

The algorithm defines the O set of operations

$$O = \{\beta_1(x), \beta_2(x), \beta_3(x), \beta_4(x), \beta_5(x), \beta_6(x)\} \tag{4}$$

where

- $\beta_1(x)$ is writing (reading) 0, if x is even and 1 if x is odd,
- $\beta_2(x)$ is writing (reading) 1, if x is even and 0 if x is odd,
- $\beta_3(x)$ is a shift to the next file in the file set, if x is even and to the previous file in the file set, if x is odd,
- $\beta_4(x)$ is a shift to the previous file in the file set, if x is even and to the next file in the file set, if x is odd,
- $\beta_5(x)$ is a shift ahead in the file by 1 fragment, if x is even and by 2 fragments, if x is odd,
- $\beta_6(x)$ is a shift ahead in the file by 2 fragments, if x is even and by 1 fragment, if x is odd.

Some of the operations are aimed at writing/reading bits of the concealed information into/from the fragment parameters. Other operations serve the purpose of determining the position shifts between the files and fragments at the time of performing the respective steps of the algorithm for the purpose of writing and/or reading confidential information. The set of operations – similarly to the set of fragment parameters – is not complete and currently still a proof of concept.

For the purposes of the proposed algorithm, the Carthesian product of the sets $O \times f_z$ where $z \in \langle 0, y-1\rangle$, $z \in \mathbb{N}^0$ allows the creation of the R set of encoding rules. Since neither the set of operations, nor the set of fragment parameters are not fully defined, similarly, the set R is not final either and currently still the proof of concept:

$$R = \{\beta_1(\alpha_1), \beta_1(\alpha_2), \beta_1(\alpha_3), \ldots, \beta_6(\alpha_3), \beta_6(\alpha_4), \beta_6(\alpha_5)\}. \tag{5}$$

The encoding rule $\beta_1(\alpha_1)$ may be then interpreted as the application of the operation $\beta_1(x) \in O$, where x is the $\alpha_1 \ \in f_z : z \in \langle 0, y-1\rangle$, $z \in \mathbb{N}^0$.

The interpretation of the encoding rules is different when hiding and extracting confidential information. For example, to the encoding rule $\beta_1(\alpha_1)$ the following applies:

> When writing confidential information, this encoding rule must ensure that the 0 bit is written – after the application of the encoding rule – by setting the address of the first cluster of the fragment used for writing the bit even, and that the 1 bit is written – after the application of the encoding rule – by

setting the address of the first cluster of the fragment used for writing the bit to an odd value.

If the current value of this fragment parameter does not correspond to the value required by the rule, the value of this fragment parameter shall be changed to the appropriate value.

When reading the hidden information, the application of this encoding rule does not change the value of the given fragment parameter. Depending on whether the fragment parameter is even/odd, a 0/1 value is read as the part of the hidden message.

3.1 Information Hiding

To apply the algorithm hiding the confidential message M represented by a final stream of bits into the fragmentation of the files stored in the file system, we have to know the sets V, A, $\forall F_n \in V$ and $\forall f_z \in A$, characterising the current state of the specific file system, into which the information shall be hidden. Moreover, also the above sets O and R have to be known and the input parameters of information hiding have to be known as the ordered set H:

$$H = \{V', R', f_{(s)}\} \tag{6}$$

where

- V' is an ordered set of files and $V' \subset V$,
- R' is an ordered set of encoding rules and $R' \subset R$,
- f_s is a fragment of a file, for which $\exists F_n \in V' : f_s \in F_n$.

Thus, from the V set of files we have to select certain files and create the subset V', into which the hidden information shall be encoded and order them according to the required order. Then, the R' subset of information encoding rules has to be selected from the R set of rules; this subset has to be ordered, too. Finally, the f_s starting fragment has to be selected, from which the encoding shall be performed. This fragment must belong to one of the selected files, though it need not be the first fragment of the given file. The information hiding parameters included in the set H form a steganographic key, used to store the concealed message – it has to be known also to extract the hidden information.

The information hiding algorithm consists of the following steps:

Step 1. Files of the set V', into which the confidential information shall be hidden, have to be ordered by their selected feature or by any permutation of their order; this order is one of the elements of the steganographic key. After ordering the files, each file shall be assigned an identifier: $fi \in \langle 0; x' - 1\rangle$, where $x' = |V'|$.

Step 2. For $\forall F_n \in V'$, its file fragments have to be ordered in the order of accessing them when reading the file. Fragment f_n is the successor of fragment f_m, if

$\exists\alpha_5 \in f_m \wedge \exists\alpha_1 \in f_n : \alpha_5 = \alpha_1$. Next, to each fragment, an ordinal number has to be assigned: $fr \in \langle 0; d-1\rangle$, where d is the number of fragments of the file F_n.

Step 3. In this step, the identifier fi of the file containing the starting fragment f_s and the fr ordinal number of the fragment in the corresponding file is found; these are registered as fi_a and fr_a, if $fi_a = fi$ and $fr_a = fr$ of starting fragment f_s. The variables fi_a and fr_a will later serve as pointers to the location of the currently processed fragment.

Step 4. In this step, the pointer m_a pointing at the current bit of the string M to be encoded is set to 0. Thus, it points to the first bit of the string M.

Step 5. To the current fragment, i.e. the one, to which the pointers fi_a and fr_a point to, we apply all encoding rules of the set R' – being part of the key – one by one, in the specified order (this order is important, because one can encode multiple bits of the concealed information using multiple rules into a single fragment, therefore the order of their encoding into the fragment must be known). If the specific rule serves for writing a bit into the corresponding fragment feature, the bit shall be written and the pointer m_a shall be incremented by 1. If, upon application of any of the rules, the last bit of the confidential message is encoded, the execution of the algorithm ends.

Step 6. The application of rules in Step 5, leading to the change of some fragment parameters requires an additional compensation of this change in some other fragment of the particular file to make sure that the information stored in the file is not corrupted and to prevent the corruption of the part of the hidden message M, already encoded in the set V'. An example may be a change of the length of the specific fragment – a lengthening by one cluster requires a shortening of another fragment of the same file by a cluster. Subsequently, $\forall f_z \in A$ which is representing changed fragment must be updated to store all changes of the file fragment parameters.

Step 7. If no rule applied in Step 5 contains information as to what shift of the current file pointer – fi_a – has to be performed, this pointer shall be incremented by 1 (shift to the next file in the set V'). If the current file pointer is set to the last file, the execution continues with the first file, thus if $fi_a = x' - 1$ then $fi_a = 0$ shall apply and vice versa: if $fi_a = 0$ and a shift backwards by a file is required, $fi_a = x' - 1$ shall be set. (This principle is applied also to the shifts within the files, performed by applying the rules specified in Step 5).

Step 8. If no rule applied in Step 5 contains any information as to what shift of the pointer to the current fragment – fr_a – has to be performed, this pointer shall be incremented by 1 (shift to the next fragment).

Step 9. Continue with Step 5.

The application of the rule storing the value of the bit of the confidential message into the corresponding fragment parameter requires setting the appropriate value

of the parameter. For example, if it is the address of the first cluster of the given fragment and it should be even but it is not, the fragment has to be shifted by a cluster towards the lower address or a higher address, depending on which is better, considering the current situation in the file system. The advantage of this approach is that – in up to 50 % of the cases – the specific fragment parameter contains the appropriate value in the given state of the file system, so it need not be changed at all.

Upon the application of the rules determining the file containing the fragment, in which the encoding performed in the next application of Step 5 of the algorithm shall happen, i.e. the rules modifying the pointer fi_a and upon the application of the rules determining the fragments count of the shift, i.e. the rules modifying the pointer fr_a, there are two alternatives. The first is to accept the current setting of the fragment of the parameter used by the specific encoding rule (i.e., leave the parameter unchanged); the second alternative is to change it (from even to odd and vice versa). The choice of accepting or modifying the parameter shall then modify the further execution of the encoding procedure. This freedom is the strength of the proposed algorithm – as far as information hiding is concerned – and it also provides sufficient flexibility to perform the least possible intervention into the current state file fragmentation during the encoding.

3.1.1 Usage Example

The following example shows a FAT file system containing four files, that are forming set V:

$$V = \{F_0, F_1, F_2, F_3\}.$$

The file system contains 16 file fragments

$$A = \{f_0, f_1, f_2, f_3, f_4, f_5, f_6, f_7, f_8, f_9, f_{10}, f_{11}, f_{12}, f_{13}, f_{14}, f_{15}\}.$$

The structure of the four respective files, which are constructed by the use of fragments from the set A, is defined by the sets F_0 to F_3:

$$\begin{aligned} F_0 &= \{f_0, f_1, f_2, f_3\}, \\ F_1 &= \{f_4, f_5, f_6, f_7\}, \\ F_2 &= \{f_8, f_9, f_{10}, f_{11}\}, \\ F_3 &= \{f_{12}, f_{13}, f_{14}, f_{15}\}. \end{aligned}$$

The parameters of the respective fragments are defined by the sets f_0 to f_{15}

$$f_0 = \{2, 4, 5, 1, 7\}, \qquad f_8 = \{55, 3, 57, 1, 59\},$$
$$f_1 = \{7, 3, 9, 1, 11\}, \qquad f_9 = \{59, 2, 60, 2, 63\},$$
$$f_2 = \{11, 2, 12, 5, 18\}, \qquad f_{10} = \{63, 1, 63, 3, 67\},$$
$$f_3 = \{18, 4, 21, 0, 0\}, \qquad f_{11} = \{67, 4, 70, 0, 0\},$$
$$f_4 = \{28, 3, 30, 1, 32\}, \qquad f_{12} = \{48, 2, 49, 1, 51\},$$
$$f_5 = \{32, 4, 35, 2, 38\}, \qquad f_{13} = \{51, 3, 53, 24, 78\},$$
$$f_6 = \{38, 2, 39, 3, 43\}, \qquad f_{14} = \{78, 4, 81, 3, 85\},$$
$$f_7 = \{43, 5, 47, 0, 0\}, \qquad f_{15} = \{85, 2, 86, 0, 0\}.$$

The binary string to be stored consists of three bits: $M =$ "011". Next, the set V', R' and the fragment $f_{(s)}$ have to be selected. The V' set of files, used to store the hidden information, was selected as F_0, F_1, F_2 in the aforementioned order:

$$V' = \{F_0, F_1, F_2\}.$$

Two encoding rules – $\beta_1(\alpha_1)$ and $\beta_3(\alpha_2)$ – were selected, forming the set R' in the aforementioned order:

$$R' = \{\beta_1(\alpha_1), \beta_3(\alpha_2)\}.$$

The selected encoding rule $\beta_1(\alpha_1)$ ensures that the bits of the confidential message shall be written to the position of the starting cluster of the fragment as follows: will it be stored at an even address in the FAT table, the bit of the encoded message shall be set to 0; will the starting cluster of the fragment be stored at an odd address in the FAT table, the bit of the encoded message shall be set to 1. According to rule $\beta_3(\alpha_2)$, if the length of the fragment is even, the next fragment should be in the file being at the next position in the V' set of files; if the length of the fragment is odd, the next fragment should be stored in the file being at the previous position in the V' set of files. No rule of the set R' specifies how many fragments should the algorithm jump when shifting after the encoding step, therefore we will select the basic shift as a shift by one fragment ahead.

As the starting fragment f_s we selected fragment f_5, being the second fragment in file F_1.

Figure 3 shows the current state of the file system. It contains four files – F_0 to F_3 – while each of the files is fragmented into four fragments. Each line contains fragments belonging to the particular file in the order of appearance in the file. The length of the rectangle representing the fragment shows its length in clusters, with the value printed just below it. Within the fragment, the figure shows its first cluster with its address in the file allocation table (FAT).

Figure 4 shows the situation after encoding the confidential message. The fragments affected by encoding – either the confidential message was encoded in their

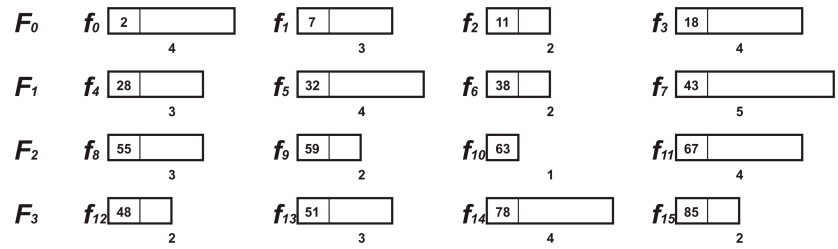


Figure 3. The current state of file fragmentation in the file system, before encoding the confidential message

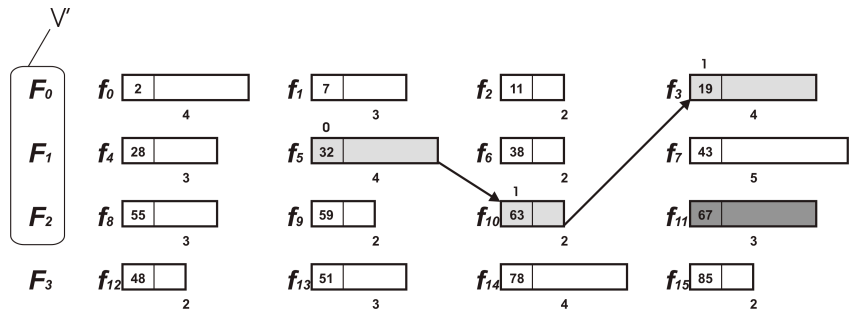


Figure 4. The confidential message $M =$ "011" stored in the fragmentation of files forming set V'

parameters, their position or their length was changed in the file system – have been set in grey.

Encoding started in fragment f_5, its first cluster is stored as an even address and the application of rule $\beta_1(\alpha_1)$ in Step 5 of the algorithm encoded the first bit of the message – bit 0 – into the fragment. The parameter of the fragment was set correctly (i.e. even), no change was needed. The following parameter of fragment f_5, used for the purpose of encoding, was the fragment length. The current fragment length is even, and the decision was taken not to change it. By applying rule $\beta_3(\alpha_2)$, the next fragment to be used for encoding was fragment f_{10}, due to the shift to the next file and next fragment.

The application of Step 5 of the algorithm to fragment f_{10} required the application of rule $\beta_1(\alpha_1)$, which encoded a further bit of the confidential message – bit 1. So it was necessary to start the fragment on an odd address in the FAT. The current setting of the parameter met the requirement, no change was necessary. We also applied rule $\beta_3(\alpha_2)$ and decided that the current value of the parameter shall be modified from odd to even. So a shift to the subsequent file was coded, however, this caused that in the next step, file F_0 became the current file (containing fragment f_3), since file F_2 was the last in the list of files in set V'. Since fragment f_{10}

was prolonged by one cluster, fragment f_{11} of file F_2 had to be shortened by one cluster to maintain the file length in clusters.

The application of Step 5 of the algorithm to fragment f_3 required the application of rule $\beta_1(\alpha_1)$. This stored another bit of the message into the fragment. This was bit 1. This required a change of the address of the first cluster of fragment f_3 to an odd value and to shift the whole fragment by a cluster. Since the last bit of the confidential message was written, the execution of the algorithm ended.

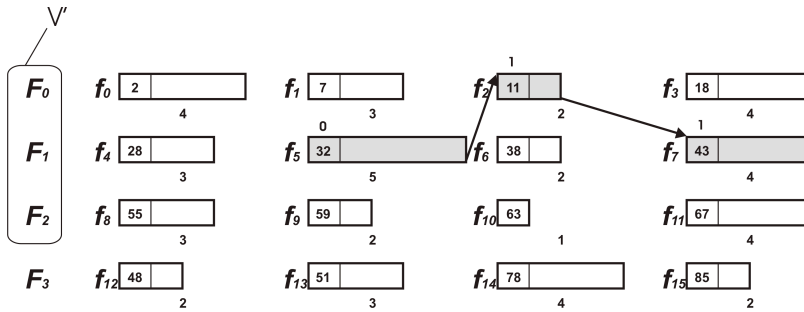


Figure 5. The confidential message $M =$ "011" stored in the fragmentation of files forming set V'

Figure 5 shows an alternative encoding of the confidential message into the fragments of files forming set V'. The length of fragment f_5 was modified to be able to continue with fragment f_2, the parameters of which were left unchanged. The next fragment, into which the information was encoded, was fragment f_7.

Its parameters were left unchanged due to the encoding of the confidential message; however, it had to be shortened by a cluster to maintain the length of file F_4, since its fragment f_5 became longer by a cluster.

3.2 Information Extraction

Information extraction is analogous to information hiding. However, one needs not know the total current state of the file system, i.e. the sets V, A, $\forall F_n \in V$ and $\forall f_z \in A$. It is enough to know the steganographic key H, i.e. the sets V', R', f_s and also $\forall F_n \in V'$ a $\forall f_z \in F_n : F_n \in V'$. So it is necessary to know the set of files, into which the confidential information was hidden, the fragments forming these files and their specific order. We also need to know the set of rules used to encode the information and also their order; moreover, one has to know the starting fragment, used to implement the encoding.

When extracting the hidden information, an algorithm with steps identical to the steps of the information hiding algorithm shall be used. When applying the rules of Step 5, we only read the bits of the message M by applying the corresponding rules and performing jumps to their respective files and fragments by applying the correct rules, without performing any changes to the file system. When reading the

message, Step 6 is not being performed. We repeatedly apply Steps 5, 7, 8 and 9, until the last bit of the hidden message M is read.

4 RESULTS AND DISCUSSION

One of the goals connected with the algorithm was to provide a flexible way of encoding the information into the parameters of the fragments of files stored in the file system. As the example in Section 3.1.1 shows, the algorithm met this requirement when it allowed encoding alternatives. Two of these are depicted in Figures 4 and 5. This allows comparison of the individual encoding alternatives, considering the number of changes of the respective fragment parameters and finding the optimum encoding, which would modify the least possible individual file fragment parameters in the current state of the file system, the specified steganographic key and the confidential message M, i.e., perform only minimal file fragmentation parameter changes.

In the example in Section 3.1.1, a single rule $\beta_3(\alpha_2)$, allowing modification of the encoding before each application in Step 5 of the proposed algorithm, was used. Each application of this rule allows the existence of two alternatives of the following encoding procedure – in case of a message of n bits it means 2^{n-1} encoding alternatives ($n-1$, since the last iteration of the application of the steps ends in Step 5, by encoding the last bit of the message, before the last application of the rule $\beta_3(\alpha_2)$. In the specific example, the number of alternative encodings is 2^2, i.e. 4, since message M is 3 bits long.

If the complete steganographic key is not specified – such as the V' set of files, on which the encoding should be implemented, is missing – during the search for the optimum encoding, the search for this set may be included in the search for the optimum encoding. In the example specified in Section 3.1.1 we may search for an appropriate permutation of the three files selected from the overall total file count (four) stored in the file system. The total number of applicable alternatives is then $(x).(x-1).(x-2)$, where $x = |V|$. Thus, the specified example allows 24 alternatives.

The use of rule $\beta_3(\alpha_2)$ without specifying set V' allows – in the example of Section 3.1.1 – a total of $(x).(x-1).(x-2).2^{n-1}$, i.e. $24.4 = 96$ encoding alternatives. To each of these alternatives, a natural number representing the number of changes to be made to the file fragment parameters to encode the specified string into the set of files using the given alternative may be assigned. Subsequently, the alternative with the lowest change count may be selected. Alternative in Figure 4 required three fragment parameter changes – in two cases, the fragment length in clusters changed and in one case, the fragment position shifted by a cluster. The alternative specified in Figure 5 required two parameter changes, with two fragment length modifications. The alternative specified in Figure 5 involved fewer file fragmentation parameter changes, so its use may be considered more advantageous.

If, during the search for the optimum encoding alternative, not even the starting fragment of the encoding is specified, the total count of encoding alternatives N_{alt} amounts to the following:

$$N_{alt} = \frac{x!}{(x - x')!} . 2^{m^{(n-1)}} . y \tag{7}$$

where

- $x = |V|$ is the number of files in the file system,
- $x' = |V'|$ is the number of files, into which the information shall be encoded,
- n is the bit count of the M confidential message,
- m is the number of rules used to modify the encoding procedure,
- y is the number of fragments of set V', where the message encoding may start.

From the above it is evident that for real-life file systems containing tens or hundreds of thousands of files and the minimum length of the confidential message being tens of bits, the search for the optimum encoding alternative is an exceptionally computing-intensive task, so it is rather a theoretical concept than a useful procedure.

However, one may search effectively for suboptimal solutions, e.g. when encoding the confidential information into the specific fragment, alternatives for defined number of applications of Steps 5 to 9 of the algorithm shall be searched for selecting the optimum encoding alternative for the given fragment and only this search window is considered. This limits the computing requirements of the procedure and allows controlling it by setting the size of the aforementioned search window.

A further design ambition related to the algorithm was to allow the least possible interference with the statistical parameters of file fragmentation in the file system and thus lower the chances of recognition of the use of this algorithm by using steganalytic methods. When encoding confidential information in the specific example set out in Section 3.1.1, we used fragment parameters α_1 and α_2, i.e. the position of the first cluster of the fragment and fragment length, checking them for being even or odd. We may also monitor the statistical values of these parameters for the set V of all files, as well as the set of files used to store the confidential information, i.e. set V'. Table 1 summarises these parameters for the situation before the encoding (Original state) of the confidential information, following the encoding by the alternative specified in Figure 4 and following the encoding by the alternative specified in Figure 5, respectively.

As it is evident from Table 1, following the encoding of the confidential information using the alternative specified in Figure 4, the statistical values of parameter α_1 change in comparison to the original state, when the number of even settings of the parameter decreases by 6.25 % (from 43.75 % to 37.50 %), and the number of odd settings of the parameter increases by 6.25 % (from 56.25 % to 62.50 %) in the set V.

		Parameter α_1				**Parameter α_2**			
		Even		**Odd**		**Even**		**Odd**	
		No.	**%**	**No.**	**%**	**No.**	**%**	**No.**	**%**
Original state	V	7	43.75	9	56.25	10	62.50	6	37.50
	V'	5	41.67	7	58.33	7	58.33	5	41.67
Encoding I	V	6	37.50	10	62.50	10	62.50	6	37.50
(Figure 4)	V'	4	33.33	8	66.67	7	58.33	5	41.67
Encoding II	V	7	43.75	9	56.25	10	62.50	6	37.50
(Figure 5)	V'	5	41.67	7	58.33	7	58.33	5	41.67

Table 1. Statistical values of parameters α_1 and α_2 for the sets V and V' before encoding confidential information and after the encoding using the two alternatives

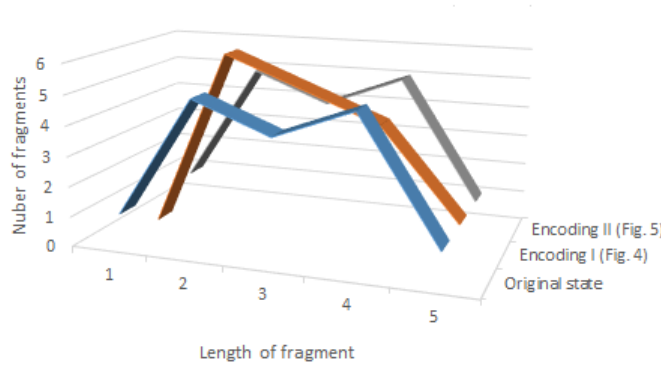
The number of even settings of the parameter decreased by 8.34 % (from 41.67 % to 33.3 %), and the number of odd settings of the parameter increased by 8.34 % (from 58.33 % to 66.67 %) in the set V'.

The statistical values of parameter α_2 remained unchanged, both in set V and also in set V'. In this aspect, the alternative specified in Figure 5 is more advantageous, since the statistical values of parameter α_1 and parameter α_2 remained unchanged, both in the case of set V as well as in the case of set V'.

Analogous to the previous case of searching the optimum encoding considering the minimum amount of fragment parameter changes, in this case we could assign each of the alternatives a value showing the degree of equality of the statistical values of the monitored parameters before and after the encoding and select the optimum alternative, thus the one with the highest degree of equality. However, the amount of encoding alternatives is identical to the previous case of searching for the optimum encoding, thus also the computing requirements of this procedure are beyond the limits of practical use.

Nevertheless, we may introduce a procedure allowing the compensation of changes to the respective fragment parameters during their execution in Step 5 of the proposed algorithm. If in the current Step 5 of the encoding a fragment with the identifiers fi_a and fr_a is selected, i.e., the fragment belonging to the file fi_a, being the at position fr_a in the file, and some of its parameters have to be changed, e.g. the parameter α_2 – its length – has to be changed from even to odd (for instance), this change may be compensated by changing the parameter from odd to even in any other fragment, to which it applies that its $fr > fr_a$ and fi may be of any permissible value, so the fragment may be part of any file of the set V'. If we add the condition of $fi = fi_a$, thus the compensating fragment (the parameter change of which serves as the compensation of the change of the fragment parameter, into which the information is encoded) and the fragment used for encoding must be from the same file, the changes of statistical values of the respective parameters shall be compensated in each individual file from the set V'.

When evaluating the statistical values of the fragment parameters, we may also consider their absolute length in clusters. The state before encoding the confiden-



Length of fragment	1	2	3	4	5
Original state	1	5	4	5	1
Encoding I (Fig. 4)	0	6	5	4	1
Encoding II (Fig. 5)	1	5	4	5	1

Original state Encoding I (Fig. 4) Encoding II (Fig. 5)

Figure 6. Distribution of fragments by their length values before encoding the confidential information and after the encoding using the first and the second alternative

tial message and after its encoding using the example in Section 3.1.1 is shown in Figure 6.

It is evident that following the encoding depicted in Figure 5, the lengths of the individual fragments are compensated to make them identical with the values they had before the encoding of the confidential information (original state). One could design an algorithm taking also the distribution of fragment lengths into account and perform compensations of these fragment parameter changes during the encoding of the information. However, practical testing performed on real-life secondary memory devices using the FAT32 file system showed that the fragment lengths were from a quite large interval – $\langle 1; 1\,862 \rangle$ – and it was not always possible to compensate change of length of one fragment by changing the length of another fragment in the way that statistical values of the set of fragments stays unchanged. Fragment length changes amounting to a single cluster do not represent significant changes in the file system as a whole, therefore we refrained from the effort to compensate the changes of this parameter.

5 CONCLUSIONS

In the introductory part, this paper analysed the current state of using secondary data storage devices, especially hard disks and the FAT file system in steganography.

Then, an algorithm using the fragmentation of a set of files stored in the FAT file system as a carrier of storing confidential information was proposed. As a proof of the concept, a set of fragment parameters, a set of available operations and a set of encoding rules were specified in the algorithm. As part of further research, these sets shall be completed with the aim to find the optimum elements. The paper also included a usage example for the information hiding algorithm and it also evaluated the computation requirements of finding the optimum encoding of the confidential information, as well as the compensation possibilities of the algorithm, considering the changes of the respective fragment parameters of the files stored within the file system.

The advantage of this algorithm is the flexibility of encoding information into a set of files, which significantly increases the complexity of extracting the confidential information using brute force attacks and allows the application of plausible deniability. The algorithm does not store any additional information on the disk and allows finding alternative encodings of the confidential information, which decreases the number of fragment parameter changes during the encoding procedure and simultaneously allows compensation of these changes to minimise the changes to the statistical values of the file fragment parameters.

A disadvantage of the algorithm is, similarly to all algorithms aiming at storing information in the fragmentation of files, the loss of information upon defragmenting the file system. This risk is limited by multiple factors. First of all, the user may forbid the process of defragmenting. The advantage of using the FAT32 file system is the possibility to use it on secondary memory media, such as pen drives, various kinds of memory cards and SSD devices, where defragmenting is suppressed due to the technology of the memory chips used. Many devices, such as mobile multimedia players, digital cameras, set-top boxes and other devices – using even traditional hard disks – often use firmware incapable of defragmenting.

Future research should focus on the development of methods allowing search of suboptimal solutions of encoding confidential information minimising the amount of interference with the file fragment parameters and procedures allowing the compensation of these changes with acceptable algorithmic complexity.

Acknowledgements

This work was supported by the Slovak Research and Development Agency under the contract No. APVV-0008-10 and KEGA 008TUKE-4/2013 Microlearning environment for education of information security specialists. The projects are being solved at the Department of Computers and Informatics, Faculty of Electrical Engineering and Informatics, Technical University of Košice. This work was supported by KEGA Agency of the Ministry of Education, Science, Research and Sport of the Slovak Republic under Grant No. 077TUKE-4/2015 "Promoting the interconnection of Computer and Software Engineering using the KPIkit". This support is very gratefully acknowledged.

REFERENCES

[1] Nag, A.—Singh, J. P.—Khan, S.—Ghosh, S.—Biswas, S.—Sarkar, D.—Sarkar, P. P.: A Weighted Location Based LSB Image Steganography Technique. In: Abraham, A., Lloret Mauri, J., Buford, J. F., Suzuki, J., Thampi, S. M. (Eds.): Advances in Computing and Communications (ACC 2011). Springer, Berlin, Heidelberg, Comunications in Computer and Information Science, Vol. 191, 2011, pp. 620–627, doi: 10.1007/978-3-642-22714-1_64. ISBN: 978-3-642-22713-4 (print), ISBN: 978-3-642-22714-1 (online).

[2] Katzenbeisser, S.—Petitcolas, F. A. P.: Information Hiding Techniques for Steganography and Digital Watermarking. Artech House Publishers, Norwood, Massachusetts, USA, 2000. ISBN: 1-58053-035-4.

[3] Nag, A.—Biswas, S.—Sarkar, D.—Sarkar, P. P.: A Novel Technique for Image Steganography Based on Block-DCT and Huffman Encoding. International Journal of Computer Science and Information Technology, Vol. 2, 2010, No. 3, pp. 103–112, doi: 10.5121/ijcsit.2010.2308.

[4] Zielińska, E.—Mazurczyk, W.—Szczypiorski, K.: Trends in Steganography. Communications of the ACM, Vol. 57, 2014, No. 3, pp. 86–95, doi: 10.1145/2566590.2566610.

[5] Nag, A.—Biswas, S.—Sarkar, D.—Sarkar, P. P.: A Novel Technique for Image Steganography Based on DWT and Huffman Encoding. International Journal of Computer Science and Security, Vol. 4, 2011, No. 6, pp. 561–570.

[6] Mazurczyk, W.—Caviglione, L.: Steganography in Modern Smartphones and Mitigation Techniques. IEEE Communications Surveys and Tutorials, Vol. 17, 2015, No. 1, pp. 334–357, doi: 10.1109/COMST.2014.2350994.

[7] Petitcolas, F. A. P.—Anderson, R. J.—Kuhn, M. G.: Information Hiding – A Survey. Proceedings of the IEEE, Vol. 87, 1999, No. 7, pp. 1062–1078, doi: 10.1109/5.771065.

[8] Anderson, R.—Needham, R.—Shamir, A.: The Steganographic File System. In: Aucsmith, D. (Ed.): Information Hiding (IH 1998). Springer, Berlin, Heidelberg, Lecture Notes in Computer Science, Vol. 1525, 1998, pp. 73–82, doi: 10.1007/3-540-49380-8_6.

[9] Morkevičius, N.—Petraitis, G.—Venčkauskas, A.—Čeponis, J.: Covert Channel for Cluster-Based File Systems Using Multiple Cover Files. Information Technology and Control, Vol. 42, 2013, No. 3, pp. 260–267, doi: 10.5755/j01.itc.42.3.3328. ISSN: 1392-124X (print), ISSN: 2335–884X (online).

[10] Kainz, O.—Jakab, F.—Michalko, M.—Feciľak, P.: Detection of Persons and Height Estimation in Video Sequence. International Journal of Engineering Sciences and Research Technology, Vol. 5, 2016, No. 3, pp. 603–609, doi: 10.5281/zenodo.48321. ISSN: 2277-9655.

[11] Ševčík, J.—Kainz, O.—Feciľak, P.—Jakab, F.: System for EKG Monitoring: Solution Based on Arduino Microcontroller. International Journal of Advanced Research in Artificial Intelligence (IJARAI), Vol. 4, 2015, No. 9, pp. 22–25. ISSN: 2165-4069.

[12] Kovalčík, M.—Feciľak, P.—Jakab, F.—Dudiak, J.—Kolcun, M.: Cost-Effective Smart Metering System for the Power Consumption Analysis of Household. International Journal of Advanced Computer Science and Applications (IJACSA), Vol. 5, 2014, No. 8, pp. 135–144, doi: 10.14569/IJACSA.2014.050821. ISSN: 2156-5570.

[13] Cheddad, A.—Condell, J.—Curran, K.—McKevitt, P.: Digital Image Steganography: Survey and Analysis of Current Methods. Signal Processing, Vol. 90, 2010, No. 3, pp. 727–752, doi: 10.1016/j.sigpro.2009.08.010.

[14] Sutherland, I.—Davies, G.—Blyth, A.: Malware and Steganography in Hard Disk Firmware. Journal in Computer Virology, Vol. 7, 2011, No. 3, pp. 215–219, doi: 10.1007/s11416-010-0149-x.

[15] Sutherland, I.—Davies, G.—Pringle, N.—Blyth, A.: The Impact of Hard Disk Firmware Steganography on Computer Forensics. Journal of Digital Forensics, Security and Law, Vol. 4, 2009, No. 2, pp. 73–84, doi: 10.15394/jdfsl.2009.1059. ISSN: 1558-7215 (print), ISSN: 1558-7223 (online).

[16] Gupta, M. R.—Hoeschele, M. D.—Rogers, M. K.: Hidden Disk Areas: HPA and DCO. International Journal of Digital Evidence, Vol. 5, 2006, No. 1, pp. 1–8.

[17] Carrier, B.: File System Forensic Analysis. Addison Wesley Professional, 2005, 600 pp., ISBN: 0-32-126817-2.

[18] Balan, C.—Vidyadharan, D. S.—Dija, S.—Thomas, K. L.: Combating Information Hiding Using Forensic Methodology. Proceedings of the Sixth International Workshop on Digital Forensics and Incident Analysis (WDFIA 2011), 2011, Kingston University, London, UK, pp. 69–75. ISBN: 978-1-84102-285-7.

[19] Aycock, J.—de Castro, D. M. N.: Permutation Steganography in FAT Filesystems. In: Shi, Y. (Ed.): Transactions on Data Hiding and Multimedia Security X. Springer, Berlin, Heidelberg, Lecture Notes in Computer Science, Vol. 8948, 2015, pp. 92–105, doi: 10.1007/978-3-662-46739-8_6.

[20] Liu, S.-F.—Pei, S.—Huang, X.-Y.—Tian, L.: File Hiding Based on FAT File System. Proceedings of the 2009 IEEE International Symposium on IT in Medicine and Education, Jinan, China, Vol. 1, 2009, pp. 1198–1201, doi: 10.1109/ITIME.2009.5236280.

[21] Sheetz, M.: Computer Forensics: An Essential Guide for Accountants, Lawyers, and Managers. John Wiley and Sons, New Jersey, USA, 2015, 176 pp., ISBN: 978-0-471-78932-1.

[22] Blaze, M.: A Cryptographic File System for Unix. Proceedings of the 1st ACM Conference on Computer and Communications Security (CCS '93), 1993, pp. 9–16, doi: 10.1145/168588.168590.

[23] Persiano, G. et al.: TCFS – Transparent Cryptographic File System. DIA, Università Degli Studi Di Salerno, Italy.

[24] Encrypting File System for Windows 2000, Microsoft Windows 2000 White Paper, Microsoft Corporation, 1998.

[25] Gutmann, P.: University of Auckland, New Zealand. The secure FileSystem (SFS) for DOS/Windows. `http://www.cs.auckland.ac.nz/pgut001/sfs/index.html`, September 1996.

[26] E4M Disk Encryption, online: http://www.e4m.net.

[27] PGPDisk, online: http://www.pgpi.org/products/pgpdisk/.

[28] Hughes, J. P.—Feist, C. J.: Architecture of the Secure File System. 2001 Eighteenth IEEE Symposium on Mass Storage Systems and Technologies, San Diego, CA, USA, 2001, pp. 277–290, doi: 10.1109/MSS.2001.10020.

[29] Hand, S.—Roscoe, T.: Mnemosyne: Peer-to-Peer Steganographic Storage. In: Druschel, P., Kaashoek, F., Rowstron, A. (Eds.): Peer-to-Peer Systems (IPTPS '02). Springer, Berlin, Heidelberg, Lecture Notes in Computer Science, Vol. 2429, 2002, pp. 130–140, doi: 10.1007/3-540-45748-8_13.

[30] Van Schaik, C.—Schmeddle, P.: A Steganographic File System Implementation for Linux. University of Cape Town, South Africa, October 1998.

[31] McDonald, A. D.—Kuhn, M. G.: StegFS: A Steganographic File System for Linux. In: Pfitzmann, A. (Ed.): Information Hiding (IH 1999). Springer, Berlin, Heidelberg, Lecture Notes in Computer Science, Vol. 1768, 2000, pp. 463–477, doi: 10.1007/10719724_32.

[32] Jókay, M.—Košdy, M.: Steganographic File System Based on JPEG Files. Tatra Mountains Mathematical Publications, Vol. 57, 2013, No. 1, pp. 65–83, doi: 10.2478/tmmp-2013-0036. ISSN: 1210-3195.

[33] Jókay, M.—Košdy, M.—Čavoj, M.: Steganographic File System Embedded in Static Images. Central European Conference on Cryptology 2013, Telč, Czech Republic, 2013, pp. 76.

Liberios Vokorokos graduated (M.Sc.) with honours at the Department of Computers and Informatics of the Faculty of Electrical Engineering and Informatics at Technical University in Košice in 1991. He defended his Ph.D. in the field of programming device and systems in 2000, his thesis title was "Diagnosis of Compound Systems Using the Data Flow Applications". He was appointed Professor for computer science and informatics in 2005. Since 1995 he has been working as an educationist at the Department of Computers and Informatics. His scientific research is focused on parallel computers of the data flow type. In addition, he also investigates the questions related to the diagnostics of complex systems. Currently he is Dean of the Faculty of Electrical Engineering and Informatics at the Technical University of Košice. His other professional interests include the membership in the Advisory Committee for Informatization at the Faculty and Advisory Board for the Development and Informatization at the Technical University of Košice.

Branislav Madoš graduated (Ing.) at the Department of Computers and Informatics at the Faculty of Electrical Engineering and Informatics of the Technical University of Košice in 2006. He defended his Ph.D. in the field of computers and computer systems in 2009, his thesis title was "Specialized Architecture of Data Flow Computer". Since 2010 he has been working as Assistant Professor at the Department of Computers and Informatics. His scientific research is focused on the parallel computer architectures and architectures of computers with data driven computational model and computer security using cryptographic and steganographic methods.

Norbert Ádám graduated (M.Sc.) with distinction at the Department of Computers and Informatics at the Faculty of Electrical Engineering and Informatics of the Technical University of Košice in 2003. He defended his Ph.D. in the field of computers and computer systems in 2007, his thesis title was "Contribution to Simulation of Feed-Forward Neural Networks on Parallel Computer Architectures". Since 2006 he has been working as Professor Assistant at the Department of Computers and Informatics. Since 2008 he is the Head of the Computer Architectures and Security Laboratory at the Department of Computers and Informatics. His scientific research is focused on the parallel computers architectures.

Anton Baláž received his Master's degree in informatics in 2004 from the Faculty of Electrical Engineering and Informatics, Technical University of Košice. In 2008 he received his Ph.D. in the area of computer security. Since 2007 he has been working as Professor Assistant at the Department of Computers and Informatics at the Faculty of Electrical Engineering and Informatics, Technical University of Košice.

Jaroslav Porubän received his M.Sc. in 2000 and his Ph.D. in computer science in 2004. Since 2013 he is the Head of the Department of Computers and Informatics at Technical University of Košice. His research is focused on the fields of empirical software engineering, domain-specific and programming languages, and human-computer interaction. He was involved in the research projects dealing with implementation of domain-specific and programming languages, language evolution and composition, and software engineering. In 2018 he established open laboratory OpenLab for evaluation of next generation human-computer interaction concepts.

Eva Chovancová graduated (Ing.) at the Department of Computers and Informatics at the Faculty of Electrical Engineering and Informatics of the Technical University of Košice in 2009. She defended her Ph.D. in the field of computers and computer systems in 2012, her thesis title was "Specialized Processor for Computing Acceleration in the Field of Computer Vision". Since 2012 she has been working as Assistant Professor at the Department of Computers and Informatics. Her scientific research is focused on the multicore computer architectures.

Computing and Informatics, Vol. 38, 2019, 367–389, doi: 10.31577/cai_2019_2_367

LIGHTWEIGHT FINGERPRINTS FOR FAST APPROXIMATE KEYWORD MATCHING USING BITWISE OPERATIONS

Aleksander Cisłak, Szymon Grabowski

Lodz University of Technology
Institute of Applied Computer Science
Al. Politechniki 11, 90–924 Łódź, Poland
e-mail: {acislak, sgrabow}@kis.p.lodz.pl

Abstract. We aim to speed up approximate keyword matching with the use of a lightweight, fixed-size block of data for each string, called a fingerprint. These work in a similar way to hash values; however, they can be also used for matching with errors. They store information regarding symbol occurrences using individual bits, and they can be compared against each other with a constant number of bitwise operations. In this way, certain strings can be deduced to be at least within the distance k from each other (using Hamming or Levenshtein distance) without performing an explicit verification. We show experimentally that for a preprocessed collection of strings, fingerprints can provide substantial speedups for $k = 1$, namely over 2.5 times for the Hamming distance and over 30 times for the Levenshtein distance. Tests were conducted on synthetic and real-world English and URL data.

Keywords: Fingerprint, keyword matching, approximate matching, bitwise

Mathematics Subject Classification 2010: 68W32

1 INTRODUCTION

This study deals with strings, that is, finite sequences of symbols. We assume that a string S is 1-indexed, i.e., index 1 refers to the first symbol $S[1]$, index 2 refers to the second symbol $S[2]$, etc. All strings are specified over the same alphabet Σ, with alphabet size $\sigma = |\Sigma|$.

Exact string comparison refers to checking whether two strings S_1 and S_2 of equal length n have the same characters at all corresponding positions. The strings can store, e.g., natural language data or DNA sequences. Assuming that each character occupies 1 byte, calculation of such a comparison takes $O(n)$ time in the worst case; however, the average case is $O(1)$, using no additional memory. Specifically, the complexity of the average case of comparing two strings depends on the alphabet size. Assuming a uniform random symbol distribution, the chance that first two symbols match (i.e., that $S_1[1] = S_2[1]$) is equal to $1/\sigma$, the chance that both first and second symbol pairs match (i.e., that $S_1[1] = S_2[1]$ and $S_1[2] = S_2[2]$) is equal to $1/\sigma^2$, etc. More generally, the probability that there is a match between all characters up to a 1-indexed position i is equal to $1/\sigma^i$.

Nonetheless, it is often faster to compare hash values for two strings (in constant time) and perform an explicit verification only when these hashes are equal to each other. This is particularly true in a situation where one would compare a single string, that is a query (pattern), against a preprocessed collection (dictionary) of strings. The hash-based approach forms the basis of, e.g., the well-known Rabin–Karp [16] algorithm for online exact matching.

Aside from exact matching, there has been a substantial interest in *approximate* string comparison, for instance for spelling suggestions or matching biological data [22, 26, 21]. Approximate string matching defines whether two strings are equal according to a specified similarity metric, and the number of errors is denoted by k in the following text. Two popular measures include:

- the Hamming distance [15] (later referred to as Ham), which defines the number of mismatching characters at corresponding positions between two strings of equal length,
- the Levenshtein distance [17] (also called edit distance, later referred to as Lev), which determines the minimum number of edits (insertions, deletions, and substitutions) required for transforming one string into another.

Hash values cannot be easily used in the approximate context. This work has focused on approximate matching in practice, and we introduce the concept of lightweight *fingerprints*, whose goal is to speed up approximate string comparison. The speedup can be achieved for preprocessed collections of strings, at the cost of a fixed-sized amount of space per each word in the collection. This means that we evaluate fingerprints for a keyword indexing problem, also known as *dictionary matching* or *keyword matching*; see, e.g., [3, 5, 6, 8, 9, 10]. Specifically, in this setting a pattern P is compared against a string collection $\mathcal{D} = \{S_1, \ldots, S_{|\mathcal{D}|}\}$. In the following, the text size is generally denoted by n, and the pattern size is denoted by m (i.e., $|P| = m$).

2 RELATED WORK

The original idea of a string fingerprint, which is also called a "sketch" in selected publications [1], goes back to the work of Rabin and Karp [28, 16]. They used a variant of a hash function called a rolling hash, which can be quickly (incrementally) calculated for each successive substring of the input text, in order to speed up exact online substring matching. This technique was later used also in the context of multiple pattern matching [20, 30] and matching over a two-dimensional text [33]. Bille et al. [4] extended this idea and demonstrated how to construct fingerprints for substrings of a string which is compressed by a context-free grammar. Policriti et al. [25] generalized the classical Rabin–Karp algorithm in order to be used with the Hamming distance.

At the conceptual level, fingerprints may be perceived as a form of lossy compression over the input text, nevertheless, they cannot replace the text – rather, they can be used as additional information. Bar-Yossef et al. [1] show that it is not possible to use only a fingerprint (reducing the text by more than a constant factor) in order to answer a match query. Moreover, they prove that for answering decision queries under the Hamming distance – such that the existence of the pattern in the text with less than k Hamming errors is reported as "a match" and no such occurrence yields the "no match" output – the size of the fingerprint must be $\Omega(n/m)$, where $k = \varepsilon m$, for a fixed $0 < \varepsilon < 1$.

Policriti and Prezza [24] presented a related idea called de Bruijn hash function, where shifting the substring by one character results in a corresponding one-character shift in its hash value. Grabowski and Raniszewski [13] used fingerprints in order to speed up verifying tentative matches in their SamSAMi (sampled suffix array with minimizers) full-text index. Fingerprints, which are concatenations of selected bits taken from a short string, allow them to reject most candidate matches without accessing the indexed text and thus avoiding many cache misses. Recently, fingerprints have been applied to the longest common extension (LCE) problem [27], allowing to solve the LCE queries in logarithmic time in essentially the same space as the input text (replacing the text with a data structure of the same size).

Ramaswamy et al. [29] described a technique called "approximate" fingerprinting; however, it refers to exact pattern matching with false positives rather than matching based on similarity metrics. Fingerprints have also been used for matching at a larger scale, i.a., for determining similarity between audio recordings [7] and files [19]. The term fingerprint has also been used with a different meaning in the domain of string processing, where it refers to the set of distinct characters contained in one of the substrings of a given string, with the ongoing recent work, e.g., a study by Belazzougui et al. [2].

3 FINGERPRINTS

In this section we introduce the notion of a fingerprint, describe its construction and demonstrate how to compare two fingerprints. For a given string S, a fingerprint S'

is constructed as $S' := f(S)$ using a function f which returns a fixed-sized block of data. In particular, for two strings S_1 and S_2, we would like to determine that $Ham(S_1, S_2) > k$ or $Lev(S_1, S_2) > k$ by comparing only fingerprints S'_1 and S'_2 for a given $k \in \mathbb{N}^+$. In other words, fingerprints allow for a quick rejection of a candidate for an approximate match (up to k errors) between two strings.

Fingerprint comparison might be indecisive, i.e., it might not be sufficient to indicate that the above stated inequalities hold. In that case (we explain later when this occurs), we still have to perform an explicit verification on S_1 and S_2, but fingerprints allow for reducing the overall number of such operations. There exists a similarity between fingerprints and hash functions; nonetheless, hash comparison works only in the context of exact matching. Let us clarify that in this work the term fingerprint refers to a short (having at most a few bytes in length) block of data which can be used for the aforementioned approximate matching.

As far as the complexity of a single verification (string comparison) is concerned, the worst case is equal to $O(n)$ for the Hamming distance (considering two strings of equal size n) and $O(k \min(|S_1|, |S_2|))$ for the Levenshtein distance, using Ukkonen's algorithm [32]. Assuming a uniform random alphabet distribution, the average case complexity is equal to $O(k)$ for both metrics.

In our proposal, fingerprints use individual bits in order to store information about symbol frequencies or positions in the string $S[1, n]$. Let $\Sigma' \subseteq \Sigma$ be a subset of the original alphabet with $\sigma' = |\Sigma'|$ denoting its size. We propose the following approaches.

- **Occurrence** (**occ** in short): we store information in each bit that indicates whether a certain symbol from Σ'_{occ} occurs in a string using σ'_{occ} bits in total.
- **Occurrence halved**: the fingerprint refers to occurrences in the first and second halves of S, that is, $S[1, \lfloor n/2 \rfloor]$ and $S[\lfloor n/2 \rfloor + 1, n]$, respectively. We store information whether each of the σ'_{occh} symbols occurs in the first half of S using the first σ'_{occh} bits of the fingerprint, and we store information whether each of the same σ'_{occh} symbols occurs in the second half of S using the second σ'_{occh} bits of the fingerprint. The occurrence halved scheme works only for the Hamming distance.
- **Count**: we store a count (i.e., the number of occurrences) of each symbol using b bits per symbol. The count can be in the range $[0, 2^b - 1]$, where $2^b - 1$ indicates that there are $2^b - 1$ or more occurrences of a given symbol. We use σ'_{count} symbols from Σ'_{count}.
- **Position** (**pos** in short): we can encode information regarding the first (leftmost, i.e., the one with the lowest index) position in S of each symbol from Σ'_{pos} using p bits per symbol, where $p \leq \lceil \log_2 n \rceil$. This position can be in the range $[1, 2^p - 1]$ encoded in the fingerprint as 0-indexed, where index 0 refers to the first symbol, index 1 refers to the second symbol, etc, and the value of $2^p - 1$ indicates that the first occurrence is either at one of the positions from the range $[2^p, n]$ or the symbol does not occur in S (we do not know which one is true). We use

$\sigma'_{pos} \cdot p$ bits in order to encode positions of σ'_{pos} symbols. The remaining bits, e.g., 1 bit for $\sigma'_{pos} = 5$, $p = 3$ and 16 bits per fingerprint, are used in order to store information about the occurrences of additional symbols, in the same fashion as in the occurrence fingerprint which was introduced previously. The position-based scheme works only for the Hamming distance.

Fingerprints can be also differentiated based on the symbols which they refer to. The choice of the specific symbol set is important when it comes to an empirical evaluation and it is discussed in more detail in Section 4. We have identified the following possibilities.

- **Common**: A set of symbols which appear most commonly in a given collection.
- **Rare**: A set of symbols which appear least commonly in a given collection.
- **Mixed**: A mixed set where half of the symbols comes from the common set while the other half comes from the rare set.

3.1 Fingerprint Examples

In the following examples, we constrain ourselves to the variant of 2-byte (16-bit) fingerprints with common letters. Fingerprints could in principle have any size, and the longer the fingerprint, the more information we can store about the character distribution in the string. Still, we regard 2 bytes, which correspond to the size of 2 characters in the original string, to be a desirable compromise between size and performance (consult the following section for experimental results). The choice of common letters is arbitrary at this point and it only serves the purpose of idea illustration.

In the following examples, occurrence fingerprint is constructed using selected 16 most common letters of the English alphabet, namely {`e`, `t`, `a`, `o`, `i`, `n`, `s`, `h`, `r`, `d`, `l`, `c`, `u`, `m`, `w`, `f`} [18, p. 36]. For the occurrence halved and count fingerprints (with $b = 2$ bits per count), we use the first 8 letters from this set. In the case of a position fingerprint (with $p = 3$ bits per letter), we use the first 5 letters for storing their positions and the sixth letter `n` for the last (single) occurrence bit.

Each fingerprint type would be as follows for the word `instance` (spaces are added only for visual presentation):

- **Occurrence**:
  `1110111000010000`
  The first (leftmost) bit corresponds to the occurrence of the letter `e` (which does occur in the word, hence it is set to `1`), the second bit corresponds to the occurrence of the letter `t`, etc.
- **Occurrence halved**:
  `01 10 01 00 10 11 10 00`
  The first (leftmost) bit corresponds to the occurrence of the letter `e` in the first half of the word, that is `inst`; the second bit corresponds to the occurrence of the

letter `e` in the second half of the word, that is `ance`; the third bit corresponds to the occurrence of the letter `t` in the first half of the word, the fourth bit corresponds to the occurrence of the letter `t` in the second half of the word, etc.

- **Count**:
  `01 01 01 00 01 11 01 00`

  For reasons which will become clear later (see proof of Theorem 1), we use a Gray code [12], in which the 2-bit encodings of numbers $\{0, 1, 2, 3\}$ are `00`, `01`, `11`, and `10`, respectively. The first two (leftmost) bits correspond to the count of the letter `e` (it occurs once, hence the count is `01`, that is 1), the second two bits correspond to the count of the letter `t` (it occurs once, hence the count is `01`, that is 1), etc.

- **Position**:
  `111 011 100 111 000 1`

  The first three (leftmost) bits correspond to the position of the first occurrence of the letter `e` (this 0-indexed position is equal to 7, hence it is set to 111), the second three bits correspond to the position of the first occurrence of the letter `t` (this 0-indexed position is equal to 3, hence it is set to `011`), etc. The last (rightmost) occurrence bit indicates the occurrence of `n`, and since this letter does occur in the input string, this bit is set to `1`.

3.2 Construction

The construction of various fingerprint types is described below. For the description of symbols and types, consult preceding subsections. At the beginning, each bit of the fingerprint is always set to `0`.

- **Occurrence**: Let us remind the reader that the length of the fingerprint is equal to σ'_{occ} for a selected alphabet Σ'_{occ} of letters whose occurrences are stored. A string is iterated characterwise. For each character c, a mask `0x1` is shifted q times to the left, where $q \in \{0, \ldots, \sigma'_{occ} - 1\}$ is a corresponding shift for the character c. In other words, there exists a mapping $c \to q$ for each character $c \in \Sigma'_{occ}$. A natural approach to this mapping is to take the position of a symbol in the alphabet Σ'_{occ} (assuming that the alphabet is ordered). The fingerprint is then `or`-ed with the mask in order to set the bit which corresponds to character c to `1`. For a string of length n, time complexity of this operation is equal to $O(n)$.
- **Occurrence halved**: The fingerprint is constructed in an analogous way to the occurrence approach described above. We start with iterating the first half of the string, setting corresponding bits depending on letter occurrences, and then we iterate the second half of the string, again setting corresponding bits, which are shifted by 1 position with respect to bits set while iterating the first half of the string. Character mapping is adapted accordingly.

- **Count**: A string is again iterated characterwise. The length of the fingerprint is equal to $b \cdot \sigma'_{count}$ for a selected alphabet Σ'_{count} of letters whose counts are stored. Similarly to the occurrence fingerprint, there exists a mapping $c \to q$ for each character $c \in \Sigma'_{count}$. However, since we need b bits in order to store a count, it holds that $q \in \{0, b, \ldots, \sigma'_{count} - b\}$, assuming that b divides σ'_{count}. A selected bit mask is set and the fingerprint is then `or`-ed with the mask in order to increase the current count of character c which is stored using b bits at positions $\{q, q+1, \ldots, q+b-1\}$.

  Instead of a natural binary encoding, we however use a Gray code, in which the encodings for any pair of successive values (e.g., 1 and 2) differ at a single bit position. To increment a b-bit field, from value i to $i+1$ (where $0 \leq i < i+1 < 2^b$), it is sufficient to extract this Gray-encoded field into a machine word W, and perform the operation $W := W \oplus (W \gg 1)$. The lowest b bits of W will then store the Gray-encoded value $i+1$. Naturally, we subsequently need to overwrite the original field with the obtained value from W. All of the above steps can be realized using a few simple bitwise operations. Assuming fixed b, for a string of length n, the time complexity of this operation is equal to $O(n)$.
- **Position**: In the case of position fingerprints, the length of the fingerprint is equal to $\sigma'_{pos} \cdot p$ for a selected alphabet Σ'_{pos} of letters whose positions are stored and a chosen constant p which indicates the number of bits per position. Here, we iterate the alphabet, and for each character $c \in \Sigma'_{pos}$ we search for the first (leftmost) occurrence of c in the string. Each position of such an occurrence is then successively encoded in the fingerprint, or the position *pos* is set to all `1`s if $pos \geq 2^p - 1$. For a string of length n, the time complexity of this operation is equal to $O(n \cdot \sigma'_{pos})$.

3.3 Comparison

We can quickly compare two *occurrence* (or occurrence halved) fingerprints by performing a binary `xor` operation and counting the number of bits which are set in the result (that is, calculating the Hamming weight, H_W). Let us note that H_W can be determined in constant time using a lookup table with $2^{8|S'|}$ entries, where $|S'|$ is the fingerprint size in bytes. We denote the fingerprint distance with F_D, and for occurrence fingerprints $F_D(S'_1, S'_2) = H_W(S'_1 \oplus S'_2)$. In other words, we count the number of mismatching character occurrences which are stored in individual bits.

However, let us note that F_D does not determine the true number of errors. For instance, for $S_1 =$ `run` and $S_2 =$ `ran`, F_D might be equal to 2 (occurrence differences for `a` and `u`) but there is still only one mismatch. On the other extreme, for two strings of length n, where each string consists of a repeated occurrence of one different symbol, F_D might be equal to 1 (or even 0, if the symbols are not included in the fingerprints), but the number of mismatches is n. In general, F_D can be used in order to provide a lower bound on the true number of errors, and the following

relation holds (the right-hand side can be calculated quickly using a lookup table, since $0 \leq F_D \leq 8|S'|$):

$$D(S_1, S_2) \geq \lceil F_D(S'_1, S'_2)/2 \rceil, D \in \{Ham, Lev\}. \quad (1)$$

This formula also holds for the *count* fingerprint. Let us observe that with the use of a Gray code, the value of F_D might be underestimated, e.g., for comparing two 4-bit counters storing values 4 and 7, we have $H_W(\texttt{0110} \oplus \texttt{0100}) = H_W(\texttt{0010}) = 1$, however, it is not overestimated. As far as the *position* fingerprint (which is relevant only to the Hamming metric) is concerned, after calculating the `xor` value, we do not compute the Hamming weight, rather, we compare each set of bits (p-gram) which describes a single position. The value of F_D is equal to the number of mismatching p-grams. Similarly to other fingerprint types, these values can be preprocessed and stored in a lookup table in order to reduce calculation time.

The relationship between the fingerprint error and the true number of errors is further explored in Theorem 1 and Theorem 2. In plain words, manipulating a single symbol in either string makes the fingerprint distance grow by at most 2. Let us note that Formula (1) follows as a direct consequence of this statement, with the round-up on the right-hand side resulting from the fact that fingerprint distance might be odd.

Theorem 1. Consider $\mathcal{F} = \{occ, count\}$ and assume a distance function $D \in \{Ham, Lev\}$. For any two strings S_1 and S_2, with their fingerprints S'_1 and S'_2, respectively, and the fingerprint distance between them $F_D(S'_1, S'_2) = f(S'_1, S'_2)$, where $f \in \mathcal{F}$, we have that for any string S_3 such that $D(S_2, S_3) = 1$, the following relation holds: $F_D(S'_1, S'_3) \leq F_D(S'_1, S'_2) + 2$.

Proof. Let us first consider the occurrence fingerprints and Hamming distance (that is $D = Ham$). For this distance, two strings must be of equal length (otherwise the distance is infinite), and we set $|S_1| = |S_2| = n$. The string S_2 can be obtained from S_1 by changing some of its $k = D(S_1, S_2)$ symbols, at positions $1 \leq i_1 < i_2 < \ldots < i_k \leq n$. Let V_0 be an initial copy of S_1 and in k successive steps we transform it into $V_1, V_2, \ldots, V_k = S_2$, by changing one of its symbols at a time. For clarity, we shall modify the symbols in the order of their occurrence in the strings (from left to right). We shall observe how the changes affect the value of $F_D(S'_1, V'_j)$, which is initially (i.e., for $j = 0$) equal to zero.

Consider a j^{th} step, for any $1 \leq j \leq k$. We have four cases:

(i) both $V_{j-1}[i_j] \in \Sigma'$ and $V_j[i_j] \in \Sigma'$,

(ii) both $V_{j-1}[i_j] \notin \Sigma'$ and $V_j[i_j] \notin \Sigma'$,

(iii) $V_{j-1}[i_j] \in \Sigma'$ but $V_j[i_j] \notin \Sigma'$,

(iv) $V_{j-1}[i_j] \notin \Sigma'$ but $V_j[i_j] \in \Sigma'$.

Let us notice that:

- in case (i) $H_W(V'_j) - H_W(V'_{j-1}) \in \{-1, 0, 1\}$, yet since $V_{j-1}[i_j] \neq V_j[i_j]$, we may obtain new mismatches at (at most) two positions of the fingerprints, i.e., $F_D(S'_1, V'_j) - F_D(S'_1, V'_{j-1}) \leq 2$,
- in case (ii) $V'_j = V'_{j-1}$ and thus $F_D(S'_1, V'_j) = F_D(S'_1, V'_{j-1})$,
- in case (iii) $H_W(V'_{j-1}) - H_W(V'_j) \in \{0, 1\}$ and $F_D(S'_1, V'_j) - F_D(S'_1, V'_{j-1}) \leq 1$,
- in case (iv) $H_W(V'_j) - H_W(V'_{j-1}) \in \{0, 1\}$ and $F_D(S'_1, V'_j) - F_D(S'_1, V'_{j-1}) \leq 1$.

From the shown cases and by the triangle inequality we conclude that replacing a symbol with another makes the fingerprint distance grow by at most 2.

Now we change the distance measure to the Levenshtein metric (i.e., we set $D = Lev$). Note that the set of available operations transforming one string into another is extended; not only substitutions are allowed, but also insertions and deletions. The overall reasoning follows the case of Hamming distance, yet we need to consider all three operations. A single substitution in V_j, for a j^{th} step, makes the fingerprint distance grow by at most 2, in the same manner as shown above for the Hamming distance. Inserting a symbol c into V_j (at any position) implies one of three following cases:

(i) $c \notin \Sigma'$, where the fingerprint distance remains unchanged,

(ii) $c \in \Sigma'$ and $c \in S_1$, where again the fingerprint distance does not change, or

(iii) $c \in \Sigma'$ and $c \notin S_1$, where the fingerprint distance grows by 1.

Deleting a symbol c from V_j (at any position) implies one of three following cases:

(i) $c \notin \Sigma'$, where the fingerprint distance remains unchanged (same as for the insert operation),

(ii) $c \in \Sigma'$ and $c \in S_1$, where the fingerprint distance might not change (if V_j contains at least two copies of c) or it might grow by 1, or

(iii) $c \in \Sigma'$ and $c \notin S_1$, where the fingerprint distance might not change or it might decrease by 1. Note, however, that the last case for the delete operation never occurs in an edit script transforming S_1 into S_2 using a minimum number of Levenshtein operations.

Handling $f = count$ is analogous to the presented reasoning for $f = occ$, both for the Hamming and the Levenshtein distance. Note that changing a symbol's count by 1, where the count is stored in a b-bit field, may change up to b bits in natural binary encoding while it changes only 1 bit in Gray encoding, which is why we use the latter representation. □

Theorem 2. Consider $F_D = pos$ and assume a distance function $D = Ham$. For any two strings S_1 and S_2, with their fingerprints S'_1 and S'_2, respectively, we have that for any string S_3 such that $D(S_2, S_3) = 1$, the following relation holds: $F_D(S'_1, S'_3) \leq F_D(S'_1, S'_2) + 2$.

Proof. For the Hamming distance, two strings must be of equal length and let us set $|S_1| = |S_2| = n$. Similarly to the case of occurrence and count fingerprints, the string S_2 can be obtained from S_1 by changing some of its $k = D(S_1, S_2)$ symbols. This proof follows the same logic as presented in proof for Theorem 1. Let us note that the only difference lies in the fact that we compare the first position of a given letter rather than its occurrence. We deal with the same four cases depending on whether a modified letter belongs to Σ', and modifying a single letter may change: in case (i) at most two p-grams (which describe the position of the first occurrence of a given letter), in case (ii) 0 p-grams, in case (iii) at most one p-gram, and in case (iv) at most one p-gram (that is, the result of these two latter cases is equivalent). Again, as before, the number of modified p-grams corresponds directly to the maximum change in fingerprint distance. □

3.4 Storage

Even though the true distance is higher than the fingerprint distance F_D, fingerprints can still be used in order to speed up comparisons because certain strings will be compared (and rejected) in constant time using only a fixed number of fast bitwise operations and array lookups. As mentioned before, we consider a scenario where a number of strings is preprocessed and stored in a collection. Since the construction of a fingerprint for the query string might be time-consuming, fingerprints are useful when the number of strings in a collection is relatively high. When it comes to the space overhead incurred by the fingerprints, for a dictionary $\mathcal{D}$ containing $|\mathcal{D}|$ keywords, it is equal to $O(|\mathcal{D}||S'| + 2^{|S'|} + \sigma)$, $|S'|$ being the (constant) fingerprint size in bytes. This holds since we have to store one constant size fingerprint per keyword together with the lookup tables which are used in order to speed up fingerprint comparison. These tables include one for determining the number of mismatches between two fingerprints (depending on fingerprint type: between occurrences, between counts, etc.) and one for the resulting number of errors (see Formula (1)). Let us note that this overhead is relatively small, especially when the size of each string is large (this is further discussed in the next section).

4 EMPIRICAL STUDY

Experimental results were obtained on the machine equipped with the Intel i7-4930K processor running at 3.4 GHz and 64 GB DDR3 RAM (1.6 GHz, latency timing 9-9-9-27). The source code and a compiled Linux binary executable (using `gcc` 64-bit version 5.4.0) are publicly available under the following link: `https://github.com/MrAlexSee/Fingerprints`. Consult Appendix A for more information regarding the usage of this tool.

The following data sets were used in order to obtain the experimental results.

- **Synthetic data**: 9.0 MB, generated based on English language letter frequencies [18, p. 36], 500 000 words.

- **English insane** (real-world data): 3.18 MB, American English language dictionary, 350 518 words.
- **English 200** (real-world data): 7.41 MB, words extracted from the English 200 collection from the Pizza&Chili index (`http://pizzachili.dcc.uchile.cl/texts/nlang/english.200MB.gz`), 815 935 words. This might be regarded as a kind of a middle-ground between synthetic and real-world data. Sequences were split on any white space and they included only printable characters. These words are usually not actual English language words (they contain, e.g., punctuation marks), however, they appear as part of the English text (hence they might be searched for in practice).
- **URLs** (real-world URL data): 95.02 MB of web addresses, available online: `http://data.law.di.unimi.it/webdata/in-2004/`, 1 382 908 words.

All dictionaries were filtered in order to contain only ASCII characters (given sizes pertain to dictionaries after said filtering, not counting duplicate words or delimiters, $1\,\mathrm{MB} = 10^6\,\mathrm{B}$). All dictionaries and query collections are available directly from the Github repository mentioned above, with the exception of URLs, owing to size limitations.

The number of queries of a given size (letter count) was equal to 10 000, and the number of iterations was set to 100. Each iteration consisted in a single search for each query within the dictionary. All presented results, including searching and construction, refer to single-thread performance, measured as elapsed CPU time. For the calculation of the Hamming distance, a regular loop which compares each consecutive character until k mismatches are found was used. It turned out that this implementation was faster than any other low-level approach (e.g., directly using certain processor instructions from the SSE extension set) when full compiler optimization (level `O3`) was used. For the Levenshtein distance, we used our own implementation based on the optimal calculation of the $2k + 1$ strip and the 2-row window [14]. It turned out to be faster than publicly available implementations, for instance the version from the Edlib library [31] or the SeqAn library [11]. This was probably caused by the fact that we could use the most lightweight solution and thus omit certain layers of abstractions from the libraries, especially since the comparison function was invoked multiple times for relatively short strings.

Queries were extracted randomly from the dictionary and compared against this dictionary. We have also tried distorting the queries by inserting a number of errors. For each query, the number of errors was uniformly sampled in the range $[1, e]$, and the timing results were consistent for any e in the $[1, \ldots, 4]$ range. In the case of English language dictionaries, we have also tested queries which consisted of the most common words extracted from a large corpus of the English language, and identical behavior was observed as in the case of queries which were sampled from the dictionary. This test was performed in order to check whether the words which are more likely to be searched for in practice exhibit the same behavior as other words.

Each fingerprint occupied 2 bytes, since 1-byte fingerprints turned out to be ineffective, and we regarded this as the optimal value with respect to a reasonable keyword size. The mode length in English dictionaries was equal to 8, which means that each fingerprint roughly incurred a 25 % storage penalty on average; however, the mode length in the URL collection was equal to 69, which means that each fingerprint roughly incurred only a 3 % storage penalty on average. Count fingerprints used 2 bits per count, that is, we set $b = 2$ and a natural binary encoding (i.e., not a Gray code). For correctness, in the case of the count variant, multiple mismatches between 2-bit counters were treated as a single mismatch. This was the case since, e.g., the difference between 1 and 2 is equal to 1, but $\texttt{01} \oplus \texttt{10} = \texttt{11}$ (the Hamming weight of such result is equal to 2 and not to 1). Position fingerprints used 3 bits per position, that is, we set $p = 3$ (consult Section 3 for details). Given the selected fingerprint size of 2 bytes (16 bits), these values allow for the use of 8 letters for count fingerprints and 5 letters for position fingerprints, with an extra occurrence bit in the latter case.

In our implementation, fingerprint comparison requires performing one bitwise operation and 2 array lookups, that is, 3 constant operations in total. We analyze the comparison time between two strings using various fingerprint types versus an explicit verification. When the fingerprint comparison was not decisive (i.e., we could not reject the match based solely on the use of fingerprints), a verification consisting in distance calculation was performed and it contributed to the elapsed time. The fingerprint is calculated once per query and it is then reused for the comparison with consecutive keywords. This means that we examine the situation where a single query is compared against a set (dictionary) of keywords.

4.1 Results

Figure 1 demonstrates the results for synthetic English data, which allowed us to check a wide range of word sizes (which occur infrequently or not at all in natural language corpora) for occurrence, count, and position fingerprints. Hamming distance was used as a similarity metric in this case. As described in the previous section, common, mixed, and rare letter sets were selected based on English alphabet letter frequencies. We can observe that the effectiveness of various approaches depends substantially on the word size, and the performance of letter sets also depends on the fingerprint type. The highest speedup was provided by occurrence fingerprints for common letters in the case of words of 10 characters, and it was equal to over 2.5 times with respect to the naive comparison.

In Tables 1 and 2 we present the speedup for $k = 1$ that was achieved for two English dictionaries and the URL data. Word lengths of 8 (in total 48 636 words for English insane and 104 753 words for English 200) and 69 (in total 34 044 words for URLs) were used, which corresponded to mode length values in the tested dictionaries. The speedup S was calculated using the following formula: $S = T_n/T_f$, where T_n refers to the average time required for a naive comparison (i.e., not using fingerprints), and T_f refers to the average time required for comparison using fin-

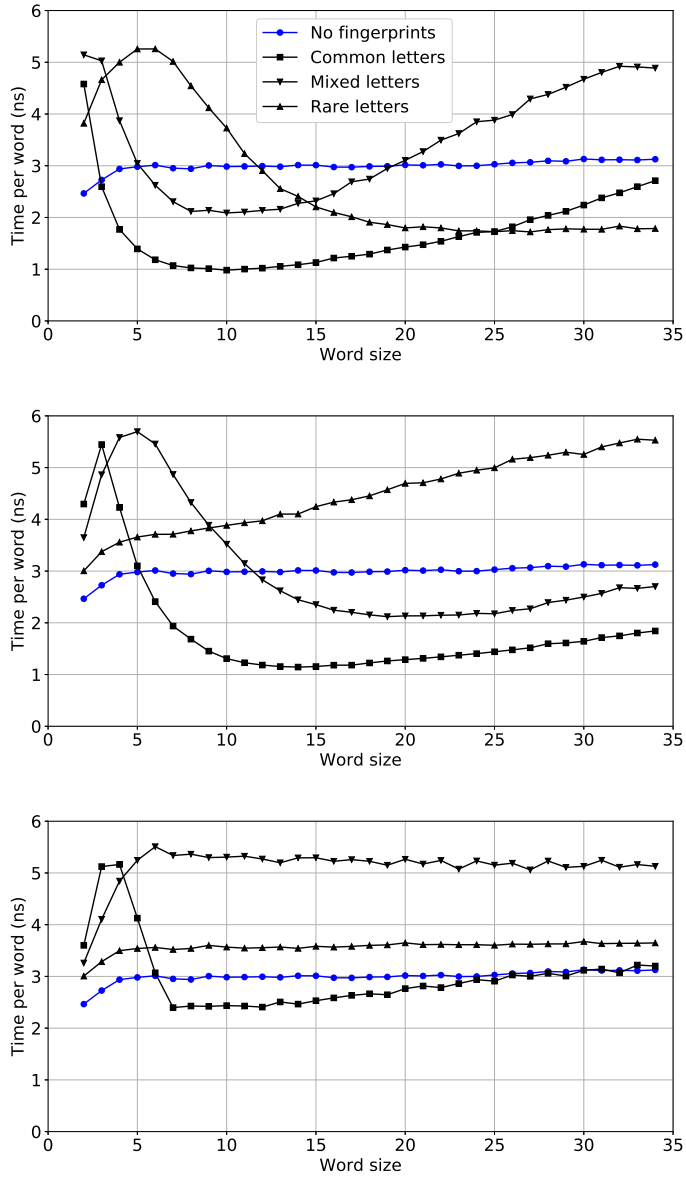


Figure 1. Comparison time vs. word size for 1 mismatch (**Hamming** distance) for **synthetic** data. Words were generated over the English alphabet. Time refers to average comparison time between a single pair of words. The upper figure shows results for **occurrence** fingerprints, the middle figure shows results for **count** fingerprints, and the bottom figure shows results for **position** fingerprints.

gerprints. For instance, 2.0 means that the time required for comparison decreased twofold when fingerprints were used.

A higher speedup in certain cases for the URL data was caused by a higher level of similarity between the data. In particular, the data set comprised some URLs which referred to different resources that were located on the same server. This resulted in certain words sharing a common prefix, requiring a naive algorithm to proceed with checking at least several first characters of each word. Presented results also demonstrate a limitation of our technique, which is apparent in the case of shorter words, where using fingerprints may increase the comparison time. Position fingerprints are not listed for the URL data, since they were completely ineffective due to multiple common prefixes between words and a large word length (almost no words were rejected). Let us also note that Hamming distance results for the English words are consistent with those reported for synthetic English data.

English Insane	Common	Mixed	Rare
Occurrence	2.66	1.45	0.72
Occurrence halved	2.05	0.97	0.69
Count	1.26	0.64	0.72
Position	1.03	0.55	0.80

English 200	Common	Mixed	Rare
Occurrence	2.12	1.03	0.54
Occurrence halved	1.35	0.61	0.75
Count	0.88	0.50	0.77
Position	0.71	0.52	0.82

URLs	Common	Mixed	Rare
Occurrence	1.46	1.19	2.27
Occurrence halved	1.78	1.93	1.53
Count	1.82	1.78	1.38

Table 1. Speedup for various fingerprint types relative to a naive comparison for $k = 1$ using **Hamming** distance for **real-world** data (English and URL dictionaries). Values smaller than 1.0 indicate that there was no speedup and the time required for comparison increased. The results in upper and middle table were calculated for the set of English language words of length 8, and the results in the lower table were calculated for the set of URLs of length 69 (both length values were modes of the word lengths in the respective dictionaries).

In Table 3 we list percentages of words that were rejected for the same data sets for $k = 1$ as a hardware-independent method of comparing our approaches. The rejection rate is naturally positively correlated with the speedup in comparison time. Table 4 presents the construction speed. Time measured during the construction covered (i) creating fingerprints, (ii) storing fingerprints in our custom dynamic

container, (iii) storing words from the input dictionary (not counting disk I/O) in the same container. It is assumed that the words in the dictionary are already sorted, a stage which can be easily performed as preprocessing (most available dictionaries are already sorted anyway).

English Insane	Common	Mixed	Rare
Occurrence	33.38	10.19	3.44
Count	8.34	2.95	1.05

English 200	Common	Mixed	Rare
Occurrence	22.62	6.50	1.98
Count	5.11	1.98	1.01

URLs	Common	Mixed	Rare
Occurrence	3.60	2.19	14.35
Count	5.52	5.87	3.06

Table 2. Speedup for various fingerprint types relative to a naive comparison for $k = 1$ using **Levenshtein** distance for **real-world** data (English and URL dictionaries). The results in upper and middle table were calculated for the set of English language words of length 8, and the results in the lower table were calculated for the set of URLs of length 69.

Let us also discuss a related method, namely neighborhood (permutation) generation [23]. For a given pattern P, it consists in constructing all combinations of perturbed words derived from P, whose presence in the dictionary is then checked in an exact manner (using, e.g., a hash table). For instance, using Hamming distance for a word `cat` and English alphabet, one would first check a perturbation using letter `a`: `aat`, `cat` (can be ignored, since it is the same as the pattern), and `caa`, then using letter `b`: `bat`, `cbt`, `cab`, etc.

If the neighborhood size (that is, the count of generated candidates), which depends on the pattern length and the alphabet size, is relatively small and the dictionary size (that is, the total word count) is relatively large, this might turn out to be a promising approach. On the other hand, fingerprints are a more versatile method, which can be used for speeding up a comparison of any two strings. This stands in contrast to only checking for the presence of a string in a dictionary (as is the case for the neighborhood method), and fingerprints can be used for augmenting another data structure (see Section 5 for more information).

For the comparison of neighborhood generation and a fingerprint-based search when querying a dictionary, consult Figure 2. The search was performed for Hamming and Levenshtein metrics with $k = 1$. Time refers to average comparison time between a single pair of words, in the same manner as for fingerprint comparison presented in Figure 1. Both dictionaries were subsampled in order to contain the tested number of words, namely $[2^6, 2^7, \ldots, 2^{16}]$. This consisted in randomly selecting words of size 8, which was the mode length in both dictionaries.

English Insane	Common	Mixed	Rare
Occurrence (Ham, Lev)	98.45 %	92.53 %	75.84 %
Occurrence halved (Ham)	96.72 %	85.30 %	10.12 %
Count (Ham, Lev)	90.55 %	71.37 %	7.01 %
Position (Ham)	87.80 %	50.30 %	0.65 %

English 200	Common	Mixed	Rare
Occurrence (Ham, Lev)	97.22 %	87.73 %	55.07 %
Occurrence halved (Ham)	92.34 %	71.39 %	3.17 %
Count (Ham, Lev)	84.02 %	55.30 %	2.19 %
Position (Ham)	78.47 %	39 42 %	0.23 %

URLs	Common	Mixed	Rare
Occurrence (Ham, Lev)	71.00 %	55.72 %	89.39 %
Occurrence halved (Ham)	80.33 %	84.03 %	73.41 %
Count (Ham, Lev)	80.87 %	80.55 %	67.33 %

Table 3. Percentage of **rejected words** for various fingerprint types for $k = 1$ for **real-world** data (English and URL dictionaries). Rejection means that the true error was determined to be more than k based only on fingerprint comparison. The results in upper and middle table were calculated for the set of English language words of length 8, and the results in the lower table were calculated for the set of URLs of length 69.

English Insane	Common	Mixed	Rare
Occurrence	498.56	499.10	497.62
Occurrence halved	475.70	475.18	474.26
Count	155.14	165.36	316.29
Position	154.35	167.62	196.97

English 200	Common	Mixed	Rare
Occurrence	485.61	485.76	485.61
Occurrence halved	458.59	458.68	461.25
Count	165.13	190.57	335.54
Position	160.57	170.90	198.26

URLs	Common	Mixed	Rare
Occurrence	418.75	419.07	419.05
Occurrence halved	405.91	406.03	405.93
Count	244.99	270.82	357.00

Table 4. **Construction speed** given in MB/s ($1\,\text{MB} = 10^6\,\text{B}$) for various fingerprint types for **real-world** data (English and URL dictionaries). The results in upper and middle table were calculated for the set of English language words of length 8, and the results in the lower table were calculated for the set of URLs of length 69.

The largest value ($2^{16} = 65\,536$) was used only for the English 200 dictionary, since the English insane dictionary did not contain such number of words having 8 characters. The alphabet size is equal to 53 and 94 for English insane and English 200, respectively, and it was accordingly smaller for subsampled dictionaries (e.g., 33 and 56 characters for English dictionaries with 64 words, respectively). All these dictionaries can be inspected at the Github repository. The number of iterations for neighborhood generation was equal to 100 (the same as for the fingerprints).

We can see that fingerprints outperform the neighborhood generation method for smaller dictionaries, up to around 2 orders of magnitude for the smallest one (which is, admittedly, not likely to be used in a real-world scenario). As the dictionary size increases, the gap becomes smaller, with neighborhood generation being faster for the largest subsampled dictionaries. In regard to the Levenshtein distance, on the one hand there exist more combinations which need to be checked by the permutation algorithm (insertions, deletions), on the other hand, Levenshtein distance is also more expensive to calculate when the fingerprint comparison is not decisive. Nevertheless, the neighborhood generation algorithm was slower for the Levenshtein metric when compared to the Hamming metric very roughly by a factor of 5, and fingerprints were relatively slower very roughly by a factor of 2. This meant that for dictionaries where the fingerprint-based approach turned out to be faster, it outperformed neighborhood generation by a wider margin for the Levenshtein than for the Hamming distance.

Let us note that the time required to extract the alphabet from the dictionary (which was needed for generating the neighborhood) did not contribute to the measured elapsed time. The tested implementation of the neighborhood generation method can be also found in the Github repository referenced previously.

In general, the choice of the optimal strategy, viz. fingerprint type, letters data set, and how many bits are used per single counter or position in a fingerprint, depends chiefly on the input data. Larger fingerprints would allow for obtaining a better rejection rate, but this would come at the cost of increased space usage. Once the rejection rate is close to the optimal 100 %, larger fingerprints would provide only a negligible reduction in processing time. In our case, the simplest approach, that is, occurrence fingerprints with common letters, seemed to offer the best performance. Still, we would like to point out that a practical evaluation on a specific data set would be advised in a real-world scenario.

5 CONCLUSIONS

We have evaluated fingerprints in the context of dictionary matching. Still, we would like to emphasize the fact that fingerprints are not a data structure in itself, rather, they are a string augmentation technique which we believe may prove useful in various applications. For instance, they can be used in any data structure which performs multiple internal approximate string comparisons, providing

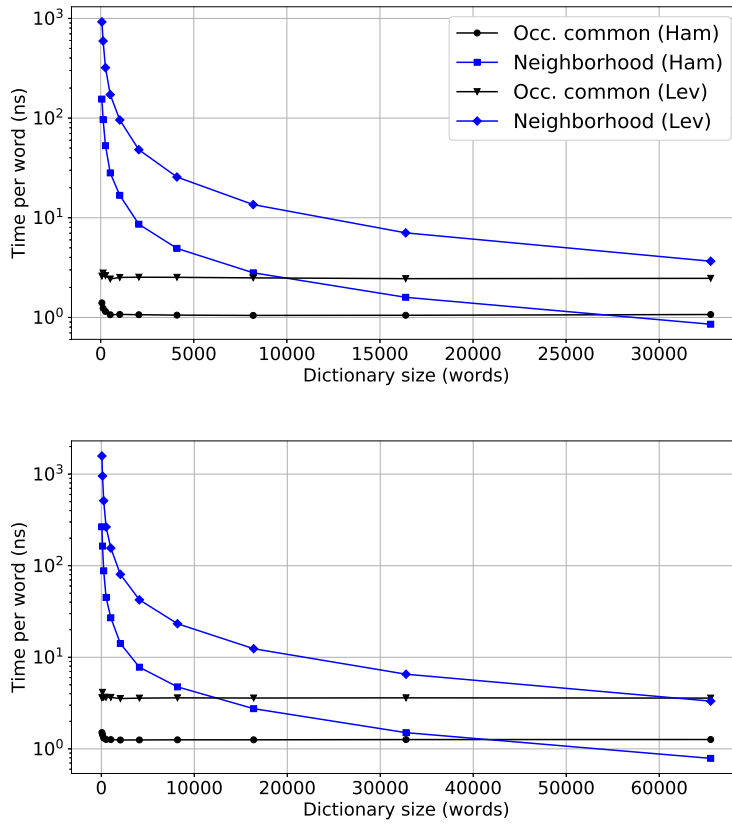


Figure 2. Comparison time vs. dictionary size for 1 error using **fingerprints** (occurrence common variant, which turned out to be the fastest one) and the related **neighborhood generation** method. Results for the English insane dictionary (upper figure) and the English 200 dictionary (lower figure) are presented, both for words of size 8. Note the logarithmic y-scale.

substantial speedups at a modest increase in the occupied space. In particular, for longer strings such as URL sequences the space overhead can be considered negligible.

Fingerprints take advantage of the letter distribution, and for this reason they were not effective for strings sampled over the alphabet with a uniform random distribution. They are also not recommended for the DNA data due to the small size of the alphabet and a large average word size. These two combined properties result in a scenario where each word contains multiple occurrences of each possible letter with a high probability.

In the future, we would like to extend the notion of a fingerprint by encoding information regarding not only single symbol distributions, but rather q-gram distributions. The set of q-grams could be determined either heuristically or using an exhaustive search, and their use might provide a speedup for any real-world data set (possibly including DNA sequences). We believe that it may be also beneficial for processing larger k values. Another possibility lies in combining different fingerprint types for a single word in order to further decrease comparison time at the cost of increased space usage. We also plan to employ fingerprints in order to speed up internal substring comparison in another data structure which we have previously created, namely the split index [9].

Acknowledgement

We thank the anonymous reviewers for their constructive comments which helped us improve the initial version of the manuscript.

A TOOL USAGE

The source code and a compiled Linux binary executable of the `fingerprints` tool are publicly available under the following link: `https://github.com/MrAlexSee/Fingerprints`. This description refers to the release version `v1.3.0` (in order to directly obtain the binary executable use the link: `https://github.com/MrAlexSee/Fingerprints/releases/download/v1.3.0/fingerprints`).

In order to reproduce experiments described in this paper, download the source code, set the path to `Boost` library in the makefile (variable `BOOST_DIR`) and issue a command `make` in the main directory. Alternatively, download directly the aforementioned compiled executable for the Linux operating system.

As mentioned in the chapter on empirical study, dictionaries and corresponding queries can be found in the `data` folder, with the exception of the URLs dictionary, which should be downloaded separately due to size restrictions (the relevant link is provided in Section 4).

In order to test the synthetic data, use the `test_synth.sh` script, and in order to test the real-world data, use the `test_real.sh` script. Both scripts automatically examine all fingerprint and letters type combinations. In order to compare fingerprints performance with a related neighborhood generation method, refer to the folder `related`. A complete list of command-line parameters which can be provided to the executable is located in Table 5.

Short Name	Long Name	Parameter Description
	`--calc-rejection`	calculate percentages of rejected words instead of measuring time
`-d`	`--dump`	dump input files and parameters info with elapsed time and throughput to output file (useful for testing)
	`--dump-construction`	dump fingerprint construction time
`-D`	`--distance arg`	distance metric: `ham` (Hamming), `lev` (Levenshtein) (default = `ham`)
`-f`	`--fingerprint-type arg`	fingerprint type: `none`, `occ` (occurrence), `occhalved` (occurrence halved), `count`, `pos` (position) (default = `occ`)
`-h`	`--help`	display help message
`-i`	`--in-dict-file arg`	input dictionary file path (positional argument 1)
`-I`	`--in-pattern-file arg`	input pattern file path (positional argument 2)
	`--iter arg`	number of iterations per pattern lookup (default = 1)
`-k`	`--approx arg`	perform approximate search (Hamming or Levenshtein) for k errors
`-l`	`--letters-type arg`	letters type: `common`, `mixed`, `rare` (default = `common`)
`-o`	`--out-file arg`	output file path (default = `res.txt`)
`-p`	`--pattern-count arg`	maximum number of patterns read from top of the pattern file
	`--pattern-size arg`	if set, only patterns of this size (letter count) will be read from the pattern file
`-s`	`--separator arg`	input data (dictionary and patterns) separator (default = newline)
`-v`	`--version`	display version info
`-w`	`--word-count arg`	maximum number of words read from top of the dictionary file

Table 5. A complete list of command-line parameters for the `fingerprints` tool

REFERENCES

[1] Bar-Yossef, Z.—Jayram, T. S.—Krauthgamer, R.—Kumar, R.: The Sketching Complexity of Pattern Matching. In: Jansen, K., Khanna, S., Rolim, J. D. P., Ron, D. (Eds.): Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques (RANDOM 2004, APPROX 2004). Springer, Berlin, Heidelberg, Lecture Notes in Computer Science, Vol. 3122, 2004, pp. 261–272, doi: 10.1007/978-3-540-27821-4_24.

[2] Belazzougui, D.—Kolpakov, R.—Raffinot, M.: Various Improvements to Text Fingerprinting. Journal of Discrete Algorithms, Vol. 22, 2013, pp. 1–18, doi: 10.1016/j.jda.2013.06.004.

[3] Belazzougui, D.—Venturini, R.: Compressed String Dictionary Look-Up with Edit Distance One. In: Kärkkäinen, J., Stoye, J. (Eds.): Combinatorial Pattern Matching (CPM 2012). Springer, Berlin, Heidelberg, Lecture Notes in Computer Science, Vol. 7354, 2012, pp. 280–292, doi: 10.1007/978-3-642-31265-6_23.

[4] Bille, P.—Cording, P. H.—Gørtz, I. L.—Sach, B.—Vildhøj, H. W.—Vind, S.: Fingerprints in Compressed Strings. In: Dehne, F., Solis-Oba, R., Sack, J. R. (Eds.): Algorithms and Data Structures (WADS 2013). Springer, Berlin, Heidelberg, Lecture Notes in Computer Science, Vol. 8037, 2013, pp. 146–157, doi: 10.1007/978-3-642-40104-6_13.

[5] Bocek, T.—Hunt, E.—Stiller, B.: Fast Similarity Search in Large Dictionaries. Technical Report No. ifi-2007.02, Department of Informatics, University of Zurich, 2007.

[6] Brodal, G. S.—Gasieniec, L.: Approximate Dictionary Queries. In: Hirschberg, D., Myers, G. (Eds.): Combinatorial Pattern Matching (CPM 1996). Springer, Berlin, Heidelberg, Lecture Notes in Computer Science, Vol. 1075, 1996, pp. 65–74, doi: 10.1007/3-540-61258-0_6.

[7] Cano, P.—Batlle, E.—Kalker, T.—Haitsma, J.: A Review of Audio Fingerprinting. Journal of VLSI Signal Processing Systems for Signal, Image and Video Technology, Vol. 41, 2005, No. 3, pp. 271–284, doi: 10.1007/s11265-005-4151-3.

[8] Chan, T. M.—Lewenstein, M.: Fast String Dictionary Lookup with One Error. In: Cicalese, F., Porat, E., Vaccaro, U. (Eds.): Combinatorial Pattern Matching (CPM 2015). Springer, Cham, Lecture Notes in Computer Science, Vol. 9133, 2015, pp. 114–123, doi: 10.1007/978-3-319-19929-0_10.

[9] Cisłak, A.—Grabowski, Sz.: A Practical Index for Approximate Dictionary Matching with Few Mismatches. Computing and Informatics, Vol. 36, 2017, pp. 1088–1106, doi: 10.4149/cai_2017_5_1088.

[10] Cole, R.—Gottlieb, L.-A.—Lewenstein, M.: Dictionary Matching and Indexing with Errors and Don't Cares. Proceedings of the Thirty-sixth Annual ACM Symposium on Theory of Computing, 2004, pp. 91–100, doi: 10.1145/1007352.1007374.

[11] Döring, A.—Weese, D.—Rausch, T.—Reinert, K.: SeqAn an Efficient, Generic C++ Library for Sequence Analysis. BMC Bioinformatics, Vol. 9, 2008, doi: 10.1186/1471-2105-9-11.

[12] Gray, F.: Pulse Code Communication. United States Patent Office Application, Serial No. US785697A, 1953.

[13] Grabowski, Sz.—Raniszewski, M.: Sampling the Suffix Array with Minimizers. In: Iliopoulos, C., Puglisi, S., Yilmaz, E. (Eds.): String Processing and Information Retrieval (SPIRE 2015). Springer, Cham, Lecture Notes in Computer Science, Vol. 9309, 2015, pp. 287–298, doi: 10.1007/978-3-319-23826-5_28.

[14] Gusfield, D.: Algorithms on Strings, Trees, and Sequences. Cambridge University Press, 1997, doi: 10.1017/CBO9780511574931.

[15] Hamming, R. W.: Error Detecting and Error Correcting Codes. The Bell System Technical Journal, Vol. 29, 1950, No. 2, pp. 147–160, doi: 10.1002/j.1538-7305.1950.tb00463.x.

[16] Karp, R. M.—Rabin, M. O.: Efficient Randomized Pattern-Matching Algorithms. IBM Journal of Research and Development, Vol. 31, 1987, No. 2, pp. 249–260, doi: 10.1147/rd.312.0249.

[17] Levenshtein, V. I.: Binary Codes Capable of Correcting Deletions, Insertions, and Reversals. Soviet Physics Doklady, Vol. 10, 1966, No. 8, pp. 707–710.

[18] Lewand, R. E.: Cryptological Mathematics. Mathematical Association of America, UK edition, 2000.

[19] Manber, U.: Finding Similar Files in a Large File System. Proceedings of the USENIX Winter 1994 Technical Conference (WTEC '94), San Francisco, California, 1994, pp. 1–10.

[20] Muth, R.—Manber, U.: Approximate Multiple Strings Search. In: Hirschberg, D., Myers, G. (Eds.): Combinatorial Pattern Matching (CPM 1996). Springer, Berlin, Heidelberg, Lecture Notes in Computer Science, Vol. 1075, 1996, pp. 75–86, doi: 10.1007/3-540-61258-0_7.

[21] Myers, E. W.: A Sublinear Algorithm for Approximate Keyword Searching. Algorithmica, Vol. 12, 1994, No. 4-5, pp. 345–374, doi: 10.1007/BF01185432.

[22] Navarro, G.: A Guided Tour to Approximate String Matching. ACM Computing Surveys (CSUR), Vol. 33, 2001, No. 1, pp. 31–88, doi: 10.1145/375360.375365.

[23] Navarro, G.—Baeza-Yates, R. A.—Sutinen, E.—Tarhio, J.: Indexing Methods for Approximate String Matching. IEEE Data Engineering Bulletin, Vol. 24, 2001, No. 4, pp. 19–27.

[24] Policriti, A.—Prezza, N.: Hashing and Indexing: Succinct DataStructures and Smoothed Analysis. In: Ahn, H. K., Shin, C. S. (Eds.): Algorithms and Computation (ISAAC 2014). Springer, Cham, Lecture Notes in Computer Science, Vol. 8889, 2014, pp. 157–168, doi: 10.1007/978-3-319-13075-0_13.

[25] Policriti, A.—Tomescu, A. I.—Vezzi, F.: A Randomized Numerical Aligner (rNA). Journal of Computer and System Sciences, Vol. 78, 2012, No. 6, pp. 1868–1882, doi: 10.1016/j.jcss.2011.12.007.

[26] Pollock, J. J.—Zamora, A.: Automatic Spelling Correction in Scientific and Scholarly Text. Communications of the ACM, Vol. 27, 1984, No. 4, pp. 358–368, doi: 10.1145/358027.358048.

[27] Prezza, N.: In-Place Sparse Suffix Sorting. Proceedings of the 29th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA '18), SIAM, 2018, pp. 1496–1508, doi: 10.1137/1.9781611975031.98.

[28] Rabin, M. O.: Fingerprinting by Random Polynomials. Technical Report TR-15-81, Department of Computer Science, Harvard University, 1981.

[29] Ramaswamy, R.—Kencl, L.—Iannaccone, G.: Approximate Fingerprinting to Accelerate Pattern Matching. 6th ACM SIGCOMM Conference on Internet Measurement (IMC '06), ACM, 2006, pp. 301–306, doi: 10.1145/1177080.1177120.

[30] Salmela, L.—Tarhio, J.—Kytöjoki, J.: Multipattern String Matching with Q-Grams. Journal of Experimental Algorithmics, Vol. 11, 2006, Art. No. 1.1, doi: 10.1145/1187436.1187438.

[31] Šošić, M.—Šikić, M.: Edlib: A C/C++ Library for Fast, Exact Sequence Alignment Using Edit Distance. Bioinformatics, Vol. 33, 2017, No. 9, pp. 1394–1395, doi: 10.1093/bioinformatics/btw753.

[32] Ukkonen, E.: Algorithms for Approximate String Matching. Information and Control, Vol. 64, 1985, No. 1-3, pp. 100–118, doi: 10.1016/S0019-9958(85)80046-2.

[33] Zhu, R. F.—Takaoka, T.: A Technique for Two-Dimensional Pattern Matching. Communications of the ACM, Vol. 32, 1989, No. 9, pp. 1110–1120, doi: 10.1145/66451.66459.

Aleksander Cisłak received his B.Sc. degree in computer science from Lodz University of Technology in 2014, M.Sc. degree in informatics from TU München in 2015, and Ph.D. degree in computer science from Warsaw University of Technology in 2019. His research interests include string matching algorithms and applied graph theory. He worked as Assistant at TU München, conducting research in the area of graph-based behavioral malware detection, and he currently works as Assistant at Warsaw University of Technology, with the main focus of his work being the fuzzy cognitive maps (FCMs).

Szymon Grabowski received his M.Sc. degree from the University of Lodz in 1996, Ph.D. degree from AGH-UST in Cracow in 2003, and Habilitation degree from Systems Research Institute in Warsaw in 2011, all in computer science. His former research, including his Ph.D. dissertation, involved nearest neighbor classification methods in pattern recognition, also with applications in image processing. Currently, his main interests are focused on string matching and text indexing algorithms, and data compression. Some of his particular research topics include various approximate string matching problems, compressed text indexes, and XML compression. He has published about 100 papers in journals and conferences. He is currently Professor at the Institute of Applied Computer Science of Lodz University of Technology.

Computing and Informatics, Vol. 38, 2019, 390–420, doi: 10.31577/cai_2019_2_390

EFFICIENT METHODS FOR SCHEDULING JOBS IN A SIMULATION MODEL USING A MULTICORE MULTICLUSTER ARCHITECTURE

Francisca A. P. Pinto

Department of Computer Science
University of Rio Grande do Norte (UERN)
Rural University of the Semi-Arid (UFERSA)
&
Secretariat of Education of the State of Ceará (SEDUC-CE)
Fortaleza, CE, Brazil
e-mail: aparecidapradop@gmail.com

Henrique J. A. Holanda, Carla K. de M. Marques

Department of Computer Science
University of Rio Grande do Norte (UERN)
59.610-210, Mossoró, RN, Brazil
e-mail: {henriqueholanda, carla.katarina}@gmail.com

Giovanni C. Barroso

Engineering Department of Teleinformatics
Federal University of Ceará (UFC)
60455-760, Fortaleza, CE, Brazil
e-mail: gcb@fisica.ufc.br

Abstract. Over the past decade, the fast advance of network technologies, hardware and middleware, as well as software resource sophistication has contributed to the emergence of new computational models. Consequently, there was a capacity increasing for efficient and effective use of resources distributed aiming to

integrate them, in order to provide a widely distributed environment, which computational capacity could be used to solve complex computer problems. The two most challenging aspects of distributed systems are resource management and task scheduling. This work contributes to minimize such problems by i) aiming to reduce this problem through the use of migration techniques; ii) implementing a multicluster simulation environment with mechanisms for load balancing; iii) plus, the gang scheduling implementation algorithms will be analyzed through the use of metrics, in order to measure the schedulers performance in different situations. Thus, the results showed a better use of resources, implying operating costs reduction.

Keywords: Multicluster, parallel jobs, migration, gang scheduling, distributed system

Mathematics Subject Classification 2010: 68M14

1 INTRODUCTION

Over the past decade, computing platforms (Cluster, Grid and Cloud) have emerged as important computational power sources [48, 49]. Traditionally, the industry main focus has been the performance improvement of computational systems, through efficient projects increasing the components density and associated with exponential growth of data size in simulation/scientific instrumentation, storage and information published on the Internet. The computational power increase from such systems has boosted investments by Internet Service Providers, Government and Research Laboratories in computing environments, which are increasingly powerful, in order to host applications ranging from social networks to scientific workflows [44].

In such context, distributed systems arise as an interesting solution providing physical resources on demand, because it allows to add computing power of many nodes interconnected through a network to perform tasks [44]. Computer distributed systems have been used due to their important attributes, such as: efficient cost, scalability, performance and reliability [46, 48, 49]. In computational grid, there are three important aspects that should be treated: task management, tasks scheduling and resources management [1]. In particular, Grid Task Scheduling (GTS) performs an important role in the whole system, where the algorithms have a direct effect on the grid. Task scheduling in heterogeneous computing environment has proven to be an NP-complete problem [2, 3, 4, 46], and it still has attracted researchers' attention.

In order to solve this problem, many types of scheduling algorithms have been proposed for distributed environments being classified in several ways, i.e., in [6], a hierarchical classification is proposed in the tree form, which divides algorithms in the higher hierarchy into local and global. For instance, [7] defines a taxonomy for scheduling problems on grid computing platforms. Smanchat and Viriyapant [8]

have extended the grid taxonomy in order to define a scheduling problem taxonomy in cloud computing. In [9], it is defined a general taxonomy providing conceptual models for problems and solutions for scheduling which allows researchers the solution properties for scheduling in a clear way.

In addition, they presented an impact analysis of this matter in the research. Extra to the set of features presented about taxonomy, there are many heuristics for task scheduling in distributed environments. In literature, there are many scheduling algorithms [12, 13, 46, 48] which deal with different types of problems and systems. Among them, we highlighted the most traditional, such as First Come First Served (FCFS), Shortest Job First (SJF), Opportunistic Load Balancing (OLB), Minimum Execution Time (MET), Minimum Completion Time (MCT), MinMin and MaxMin.

Among existent scheduling techniques, we highlighted the scheduling group or gang scheduling or co-schedulers [14, 47], which are considered to be efficient algorithms for parallel jobs scheduling, which consist of tasks that must be allocated and executed simultaneously on different processors. These types of scheduling algorithms provide interactive response time for tasks with low execution time by means of preemption, but, as a disadvantage, cause a fragmentation by reducing the system performance [17, 19, 20, 25].

In a similar way to external fragmentation in memory, resources fragmentation occurs in a grid computing consisting of a cluster set when it cannot find a cluster that can perform job tasks simultaneously, being the total number of idle computational resources across the grid larger than this number of tasks. Fragmentation occurs in the system when it presents free processors, but job computational requirements cannot be completed, thus remaining inactive resources [21]. Resources fragmentation has been a common research topic in the past two decades. Many approaches to resources fragmentation have been developed, best-fit and task migration are the most common.

Based on the points made above, the goal of this study is to invest in reducing the fragmentation caused by scheduling group as well as in response time. Among the main contributions of this work, we can highlight:

1. Heuristics implementation, Adapted First Come First Served (AFCFS), Largest Slowdown First (LXF) and Largest Job First Served (LJFS) using gang mechanism. Based on the assumption that gang scheduling causes fragmentation in the environment, we seek to use migration mechanisms; e.g, check clusters that have available processors, analyzing which jobs have their tasks at the beginning of the queue in the latter and checking the job that has the lowest number of tasks to migrate. In addition to these migration strategies, we used mechanisms to avoid unnecessary migrations, as well as system overhead.
2. Implementation of techniques and algorithms, Join the Shortest Execution Queue (JSEQ) and Opportunistic Load Balancing (OLB) for load balancing in dispatchers, grid and local, aiming to distribute jobs for clusters, in order to reduce task waiting time, and consequently improve system efficiency. We emphasize that the JSEQ is a new proposed algorithm.

3. Based on simulator study for grids, the simulation environment composition uses a Multicore Multicluster Architecture, aiming to analyze dispatchers performance in different situations, as well as the system behavior in different contexts. This proposed environment is named as Grid Schedule Management Simulator (GSMSim).
4. Plus, the scheduling algorithms implemented will be analyzed in different contexts by metrics, Average Wait Time, Average Response Time, Loss of Capacity and Utilization, in order to measure both use of the system and its fragmentation.

This paper is structured as follows: First, the related work is presented (Section 2). Section 3 presents the proposed system model. After that, we describe the system operation (Section 4). Sections 5 and 6 present the gang scheduling and migration mechanisms. In addition, we present metrics performances, which are used to analyze the scheduler performance in different situations (Section 7). Section 8 presents the results of the simulations. Finally, we present some conclusions and motivation for future work (Section 9).

2 RELATED WORK

This work aims to invest in reducing the fragmentation caused by gang schedules [12, 20, 23, 24, 25], as well as reducing the response time of jobs. Researchers are looking for efficient mechanisms to reduce execution time, as well as improve resource utilization and hence minimize the fragmentation. The latter happens on the system when there are jobs waiting in the queue to run and there are idle processors but they still cannot perform the waiting jobs. Some works in this area are presented below.

The authors [18, 35, 25] propose migration mechanisms in order to minimize the fragmentation caused by gang schedules in these environments. They implement local and grid migration strategies in the Adapted First Come First Served (AFCFS) and Largest Gang First Served (LGFS) gang schedulers in a homogeneous cluster simulation model. In the case of local migration, it is the transfer of a task from one processor queue to another that belongs to the same cluster, and grid migration involves transferring a task from one cluster to another. In addition, they use simulation in parallel job and sequential job. The latter is composed of a single task, which takes priority at the time of allocation of the task in the resource, e.g, stopping the execution of a parallel job for its execution, thus leading to an increase in the response time of this job. The authors [19] use the migration strategies proposed by the authors [18, 35, 25] in gang schedulers AFCFS and Largest Job First Served (LJFS) [18], in a single cluster simulation model, which consists of one hundred and twenty (120) Virtual Machines (VMs). These are connected through a Dispatcher Virtual Machine (DVM) dispatch, which includes a queue for jobs that cannot be dispatched at the time of their arrival to the VMs, which is when VMs are unavailable or overloaded.

As far as minimizing the fragmentation caused by gang schedulers is concerned, this was presented in the previous work [26, 27]. In these works, we apply migration strategies in the AFCFS and LGFS [18], in a multicluster simulation environment, using a hierarchical structure of two layers, consisting of managers, Grid Dispatcher (GD) and Local Dispatcher (LD) to the allocation jobs to resources. Aiming at a more efficient load balancing, these authors considered a set of load balancing strategies: the first strategy was to introduce in the GD, before sending the jobs to the clusters, a feedback of information about the clusters loads, for more efficient load balancing; and, second strategy, we used an algorithm in LD, Join the Shortest Queue (JSQ), that applies the technique in the distribution of the tasks to the processors queue. This distribution is done according to the number of tasks in the processor queue plus the running task, that is not taking into account the execution time of the tasks. Differently from the work [26], the authors of [27] analyzed the AFCFS and LGFS algorithms in a multicluster heterogeneous simulation system in relation to the amount of resources by clusters. In addition, they used different workload sizes in the system.

Differently from the works cited above, this proposal uses a Multicore Multicluster Architecture (MCMCA) in the simulation model [43], in order to meet a larger data set demand. In this environment, the heterogeneity happens in relation to the number of resources per cluster, the resource clock rate and resource characteristics in each cluster. In addition, data consists of two different types of jobs, sequential and parallel. The latter consists of several tasks that are independent and executed simultaneously. In this work, it is considered that a sequential job is a priority task that requires only one processor for execution and the least estimated processing time compared to other jobs. Therefore, upon reaching the environment, the job is sent to the best available processor, that is not paralyzing another job for execution. In case, all processors are busy, the sequential job is sent to the queue of the processor which has the shortest runtime. This is to reduce the execution time of the jobs. In addition to the above proposals, two algorithms are introduced in the LD: Join the Shortest Execution Queue (JSEQ) and Opportunistic Load Balancing (OLB) [10], which apply techniques in the distribution of tasks to the queue of processors. We emphasize that the JSEQ is a new proposed algorithm. These algorithms are intended to reduce queuing time, as well as the response time of a job and, therefore, fragmentation. In addition, the following policies are applied for the queues scheduling: AFCFS, LJFS [19] and Largest Slowdown First (LXF) [36]. These algorithms will be implemented and adapted to the gang mechanism in order to scale the tasks of the jobs allocated in queues and implemented in the simulation environment. These policies are evaluated separately in the system in different situations, using metrics to measure both system utilization and fragmentation.

In view of this, the results (see Section 8) show (compared to other gang schedulers with and without migration, different strategies in LD and changes in the priority of a sequential job and heterogeneous workloads), that the migrating AFCFS gang scheduler presented the best results efficiently in all scenarios. Thus,

the efficiency of AFCFS with migration was confirmed, as presented by the authors [18, 35, 25, 19, 26, 27]. Different from these, we analyzed the LXF gang algorithm with and without migration in the same AFCFS and LJFS scenarios. The LXF algorithm presented a lower average response time of the jobs in relation to the LJFS algorithm. In addition, through the Loss of Capacity (LoC) metric, we evaluate the fragmentation caused by the gang algorithms (AFCFS, LXF and LJFS) in an MCMA environment. According to the results, the AFCFS and LXF algorithms cause less fragmentation in the environment.

3 SIMULATOR PROPOSAL: GSMSIM

This work proposes a multicore multicluster simulator model based on queues. A simulation methodology is applied in order to validate the model and to quantify the performance under realistic conditions (see Figure 1). The simulation system (Grid Schedule Management Simulator – GSMSim) consists of a multicore multicluster environment using a two-layer hierarchical structure. It was implemented in order to analyze schedulers performance in different situations, as well as environment behavior in different contexts. This system was implemented in the Laboratory of Research Group in Applied Computer Modeling at the Federal University of Ceará (UFC).

GSMSim model is based on queueing theory (Figure 1), which is useful for system analysis – in which conflicts occur when many entities try to simultaneously access the same resource – [28] as well as in scheduling modeling for distributed systems [29]. GSMSim is composed by managers, Grid Dispatch (GD), Local Dispatch (LD), and clusters administrators.

GD is in charge for sending sequential and parallel jobs to clusters, and LD for sending tasks belonging to the jobs in processor queues. Each LD is composed of a cluster (C_i) (i ranging from 1 to m) consisting of a multicore processor set (P_l) (l ranging from 1 to M), being $\{M, i, l, m \in N\}$. Additionally, each P_l has its own queue in the system.

In the system, there were different scenarios concerning the number of processors, machines features and quantity of clusters, in order to simulate workloads, which have jobs with multiple levels of parallelism. In this study, it is considered that a system is homogeneous when machines clock rate is equal and each C_i possesses a different quantity of processors; likewise, a system is heterogeneous when machines clock rate is different ranging from 1 500, 1 600, 1 700, 1 800, 1 900, 2 000, 2 500, 3 000, 3 500 (megahertz), randomly generated at the time of creating resources in the simulation environment. Therefore, there is heterogeneity in resources of the same cluster and, consequently, among clusters. Thus, the proposed environment can be used in different scenarios.

In the developed environment, clusters belong to an administrative domain, so that they are able to communicate with GD. Besides, the communication among processors is free contention. Hence, the communication latency is calculated as

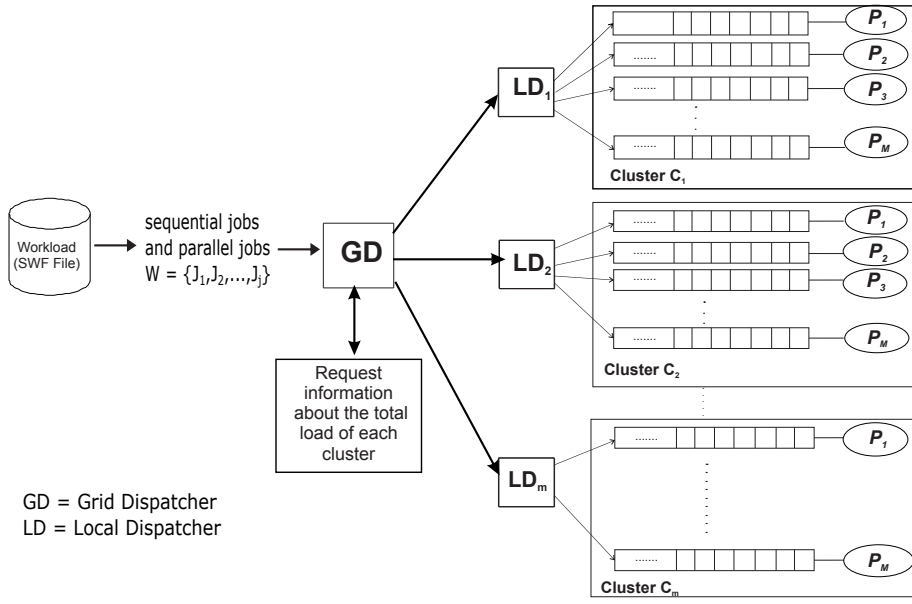


Figure 1. Multicore multicluster simulator model based on queues – GSMSim

follows [30, 31]: $T(z) = \alpha + \frac{z}{\rho}$ in which α is a constant, z represents the job size (megabytes) and ρ is the bandwidth (megabytes).

3.1 Workload

Workloads were extracted from a real distributed environment and present characteristics of Standard Workload Format (SWF) [33]. They are composed of two different types of jobs, which are competing for the same resources: sequential and parallel jobs. A workload $W = \{J_1, J_2, \ldots, J_j\}$ $(j = 1, 2, 3, \ldots)$ is composed of multiple jobs, where a job J_j is represented by a tuple (id_j, at_j, s_j, pt_j). See parameters description in Table 1.

Parameters	Description
id_j	Identification of the job, $(id_j = 1, 2, 3 \ldots)$.
at_j	Time of arrival, $(at_j \geq 0)$.
s_j	Number of tasks in a job, $(s_j \geq 1)$.
pt_j	Estimate of the processing time of a job, $(pt_j > 0)$.

Table 1. The workload parameters

A job J_j is composed of one or more tasks, e.g., $J_j = \{v_{1,j}, \ldots, v_{i,j}\}$. If $J_j = \{v_{1,j}\}$, then, $J_j = 1$ is a sequential job consisting of a single task, which requires

only one processor for its execution. Therefore, it is a high priority task, since when it arrives in the environment, it is sent to the highest speed available processor. This only happens when this task presents the shorter processing time estimated pt_j regarding to other jobs. In the case of all processors being busy, this task is sent to the processor queue that has lower execution time.

A parallel job J_j consists in $v_{j,|J_j|}$ tasks which $|J_j| > 1$. Mapping among tasks and processors must be one to one. Therefore, job tasks cannot be attributed to the same processor queue. In addition, tasks belonging to a parallel job will be scheduling for execution according to the technique scheduling in the system queue.

4 GSMSIM OPERATION

This section describes in detail the operation of system managers: GD and LD, as it is shown in Figure 1.

4.1 Grid Dispatch

GD sends jobs to clusters. This submission is based on a feedback information about the total load of each cluster, i.e., the total number of jobs in queues plus the number of tasks in execution on processors (Algorithms 1 and 2). These information about clusters load will only be sent upon a GD request, because excessive feedback may cause system overload. It is very important to know the load value of each cluster for an efficient load balancing. In case of clusters are balanced, it occurs a random dispatching.

In Equation 1, it is defined the load calculation of each cluster,

$$LC_i = \frac{1}{|P_M|} \times \sum_{p=1}^{|P_M|} [f(p) + k] \tag{1}$$

in which LC_i is the total load associated to cluster C_i (i ranging from 1 to m), $|P_M|$ is the total number of processors per cluster, $f(p)$ is the total number of tasks queued for each processor of C_i, and k represents the existence or not of a task running on processor: $k = 1$, there is a task running; otherwise, $k = 0$.

4.2 Local Dispatch

After a parallel job J_j has been sent to cluster C_i, according to the lowest workload of LC_i, LD assigns job tasks to available queues based on Opportunistic Load Balancing (OLB) algorithm, or Join The Shortest Execution Queue (JSEQ) algorithm, which were adapted and implemented in LDs.

Algorithm 1 Grid dispatcher(job)

Input: jobs
Output: T (job can be run by the system) or F (system cannot run the job)

```
1:  set S ← empty
2:  clusters_select ← 0
3:  for i = 1 to number_clusters do
4:    if (cluster[i].number_processor) < (job.number_task) then
5:      S ← S U (cluster[i])
6:      clusters_select ← clusters_select + 1
7:    else if (clusters_select > 0) then
8:      if (job.num_tasks > 1) then
9:        Cluster ← lower_load(S)
10:       Cluster.LocalDispatcher(job)
11:     else
12:       Cluster ← random(S)
13:       Cluster.LocalDispatcher(job)
14:     end if
15:   else
16:     return T
17:   end if
18:   return F
19: end for
```

4.2.1 Opportunistic Load Balancing

OLB (Algorithm 3) sends tasks belonging to a job for available processors or to their queues, regardless tasks execution time expected on processors [11]. It has the advantage of keeping machines busy but also the disadvantage of not paying due attention to about minimizing task wait time in queue, consequently, the job response time.

4.2.2 Join the Shortest Execution Queue

JSEQ algorithm, is an adjustment proposed in this work, based on Join the Shortest Queue (JSQ) [26, 27, 34]. JSEQ (Algorithm 4) is in charge for sending tasks that belong to a job for processors queues, in a way that the tasks already queued have lower execution time. It is important to notice that the execution time value sent to LD is the sum execution time of task in the queue plus the execution time of task in the processor; differently from JSQ, in which the sending of tasks to processors occurs through the quantity of tasks in processors queues. This can lead to an increase of task waiting time in queues. Thus, when a sequential job reaches the GSMSim, it has priority, as explained in Section 3.1, regardless of the algorithm that is applied in LD. Furthermore, the information feedback regarding to processors queues behavior only occurs when LD calls, thus avoiding system overload.

Algorithm 2 *lower_load*(*cluster*[*n*])

Input: set of n clusters
Output: cluster that has the lowest load

```
1: for (i = 1 to n) do
2:    if (i ≐ 1) then
3:       lower ← i
4:       size ← cluster[i].number_task() / cluster[i].number_machine()
5:    else
6:       if (cluster[i].number_task() / cluster[i].number_machine() < size) then
7:          lower ← i
8:          size ← cluster[i].number_task() / cluster[i].number_machine()
9:       end if
10:   end if
11: end for
12: return cluster(lower)
```

After distributing tasks in queues by one of the machine algorithms (OLB or JSEQ), it is used one of scheduling queues Adapted First Come First Served (AFCFS); Largest Job First Served (LJFS) [18, 19, 25] or Largest Slowdown First (LXF) [36] to scheduling tasks in queues.

The next section will present such schedulers using the gang technique adapted to task scheduling in processors queues.

Algorithm 3 OLB(Gang g)

Input: gang g

```
1: list S ← empty
2: list T ← empty
3: for i ← 1 to cluster.number_processor do
4:    if (cluster.processor[i].task_executed = null) then
5:       S.include(cluster.processor[i])
6:    else
7:       T.include(cluster.processor[i])
8:    end if
9: end for
10: sort_random(S)
11: sort_random(T)
12: cluster.processor ← empty
13: cluster.processor ← S.concatenate(T)
14: for i ← to g.number_task do
15:    cluster.processor[i].include(g.number_task[i])
16: end for
```

Algorithm 4 JSEQ(Gang g)

Input: gang g

```
1: for i ← 1 to cluster.number_processor do
2:    select ← cluster.processor[i]
3:    j ← i − 1
4:    while (j ≥ 0) and (select.time_estimated < cluster.processor[j].time_estimated) do
5:       cluster.processor[j + 1] := cluster.processor[j]
6:       j := j − 1
7:    end while
8:    cluster.processor[j + 1] ← select
9: end for
10: for i ← 1 to g.number_task do
11:    cluster.processor[i].include(g.task[i])
12: end for
```

5 GANG SCHEDULING

It is very often applied in job scheduling, in which each job is composed by a task set that must be performed simultaneously on different processors [22, 16]. This type of technique is considered efficient for scheduling parallel jobs in distributed environments, but, as a disadvantage, it results in a fragmentation reducing system performance [5, 15, 16, 19, 25].

In the simulation system, the following policies were applied for scheduling queues: Adapted First Come First Served (AFCFS), Largest Job First Served (LJFS) and Largest Slowdown First (LXF). They were adapted to gang mechanism, in order to dispatch jobs tasks allocated to queues, and implemented in a simulation environment.

5.1 AFCFS

AFCFS (Algorithm 5) tends to favor jobs with lower task number, and, consequently, requires lower number of processors. On the other hand, this may cause an increase in response time concerning larger jobs. In Algorithm 5, line 1, it is to initialize the search procedure in processors queues by jobs that have lower task number, and then, in line 6, it starts tasks exchange ordination.

Figure 2 describes the scenario where tasks belonging to jobs $J_1 = v_{1,1}, \ldots, v_{4,1}$; $J_2 = v_{1,2}, \ldots, v_{3,2}$; $J_3 = v_{1,3}, \ldots, v_{3,3}$; $J_4 = v_{1,4}, \ldots, v_{4,4}$; $J_5 = v_{1,5}, v_{2,5}$ were distributed to processors queues according to their arrival time in the system. As we can see, jobs require different quantities of processors, $J_1 = 4$; $J_2 = 3$; $J_3 = 3$; $J_4 = 4$; $J_5 = 2$, respectively. Considering the job sizes, these will be scheduled according to Algorithm 5: $J_5 = v_{1,5}, v_{2,5}$; $J_2 = v_{1,2}, \ldots, v_{3,2}$; $J_3 = v_{1,3}, \ldots, v_{3,3}$; $J_1 = v_{1,1}, \ldots, v_{4,1}$; $J_4 = v_{1,4}, \ldots, v_{4,4}$, as it is shown in Figure 3. These tasks were

Algorithm 5 AFCFS(Processor queue $p.q$)

Input: queue q of tasks of a processor p

```
1: if p.q.begin ≠ null then
2:   if start(p.q.begin) ≠ null then
3:     if p.q.begin.netx ≠ null then
4:       shorter ← p.q.begin
5:       aux ← p.q.begin.next
6:       while aux ≠ null do
7:         if aux.number_task_berlongs_job < shorter.number_task_berlongs_job
           then
8:           shorter ← aux
9:         else
10:          aux ← aux.next
11:        end if
12:      end while
13:      aux ← p.q.begin
14:      p.q.begin ← shorter
15:      p.q.begin ← aux
16:      remove_duplicate(p.q.begin)
17:    end if
18:  end if
19: end if
```

distributed in queues according to LD algorithm (OLB or JSEQ) and were then scheduled according to AFCFS policy.

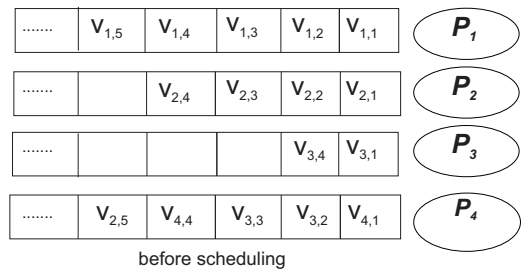


Figure 2. Jobs in queues – before scheduling

AFCFS has complexity $O(n)$, in which n is the task number in queues, which will be scheduled. It is only $O(n)$ because the scheduler passes by the queue once to check which job has the lowest number of tasks, and then forwards to the beginning of the queue where the jobs have fewer sister tasks. This situation can be performed in constant time that would be $O(1)$. Thus, complexity $O(n) + O(1) = O(n)$.

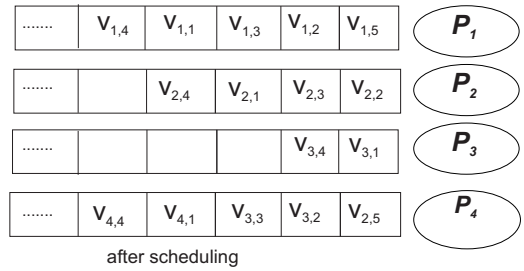


Figure 3. After scheduling – using AFCFS

5.2 LJFS

LJFS (Algorithm 6) tends to favor larger jobs performance at the expense of the smaller ones, i.e., the larger jobs have tasks allocated in processors queues before any other smaller task belonging to a job, causing an increase of response time in smaller jobs.

Algorithm 6 LJFS processor.queue $p.q$

Input: queue q of tasks of a processor p

```
1: for i ← 1 to p.q.size do
2:    select_processor ← p.q[i]
3:    i ← i − 1
4:    while (j ≥ 0) and (select.number_task_sisters > p.q[j].number_task_sisters)
      do
5:       p.q[j + 1] := p.q[j]
6:       j := j − 1
7:    end while
8:    p.q[j + 1] ← select_processor
9: end for
```

It is presented a new scenario using LJFS for the same jobs from previous example (Figure 2). Considering parallel job size (Figure 4), it will be scheduled in the following order: $J_1 = v_{1,1}, \ldots, v_{4,1}$; $J_4 = v_{1,4}, \ldots, v_{4,4}$; $J_2 = v_{1,2}, \ldots, v_{3,2}$; $J_3 = v_{1,3}, \ldots, v_{3,3}$; $J_5 = v_{1,5}, v_{2,5}$, as shown in Figure 5.

LJFS has complexity $O(n * \log(n))$, in which n is the task number in queue that will be scheduled. As the scheduler comes down to reorder the queue in descending order according to tasks number belonging to the job, this complexity is the same as a common ordering method, therefore, it is based on the merge sort ordering method [37].

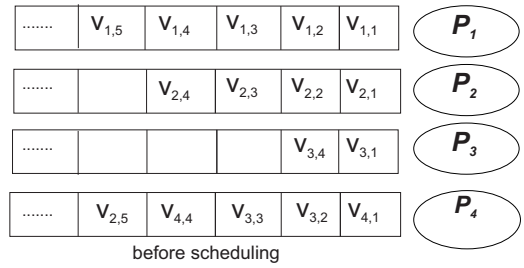


Figure 4. Jobs in queues – before scheduling

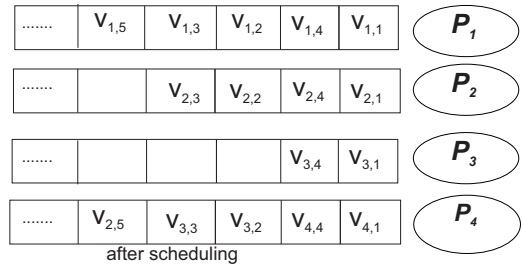


Figure 5. After scheduling – using LJFS

5.3 LXF

LXF (Algorithm 7) tends to benefit jobs that have larger Expansion Factor (XF), which is often used when comparing scheduling algorithms. It is related to metric XF, which is given by the objective function of Algorithm 7 (line 4), in which the *select.t_processing* is a job processing estimate time and *select.t_wait* is the job waiting time in the system.

Algorithm 7 LXF(Processor.queue $p.q$)

Input: queue q of tasks of a processor p

1: **for** $i \leftarrow 1$ to $p.q.size$ **do**
2: $select \leftarrow p.q[i]$
3: $i \leftarrow i - 1$
4: $t_1 \leftarrow \frac{(select.t\_processing + select.t\_wait)}{select.t\_processing}$
5: **while** $(j \geq 0)$ and $(t_1 < \frac{(p.q[j].t\_processing + p.q[j].t\_wait)}{p.q[j].t\_processing)}$ **do**
6: $p.q[j+1] := p.q[j]$
7: $j := j - 1$
8: **end while**
9: $p.q[j+1] \leftarrow select$
10: **end for**

Based on the example of Section 5.1 in Figure 7, a scenario using LXF is shown. The parallel jobs J_1, J_2, J_3, J_4 and J_5 present the following XF, respectively: 1.6; 2.5; 1.8; 1.7; 1.3. These results were calculated using the equation in line 4 (Algorithm 7), where the values of the *select.t_processing* and *select.t_wait* are collected from the jobs information, J_1, J_2, J_3, J_4 and J_5, which will be executed. Considering jobs that have larger XF, they will be scheduled in the following order $J_2 = v_{1,2}, \ldots, v_{3,2}$; $J_3 = v_{1,3}, \ldots, v_{3,3}$; $J_4 = v_{1,4}, \ldots, v_{4,4}$; $J_1 = v_{1,1}, \ldots, v_{4,1}$; $J_5 = v_{1,5}, v_{2,5}$, as shown in Figure 7.

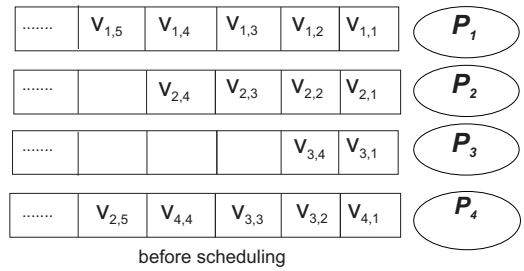


Figure 6. Jobs in queues – before scheduling

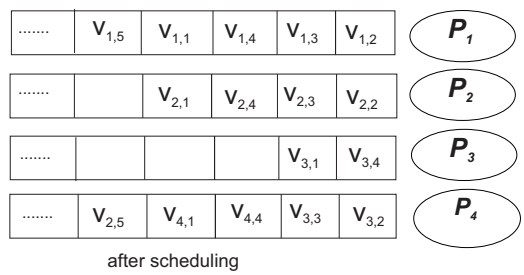


Figure 7. After scheduling – using LXF

LXF has complexity $O(n * \log(n))$, in which n is the task number in processor queue. This algorithm puts tasks in descending order, depending on the XF outcome (Section 5.3). The XF of each job is calculated in a constant time. Thus, it is considered the merge sort method as the LXF ordering algorithm.

6 MIGRATION

On the assumption that gang scheduling causes environment fragmentation, we seek to reduce fragmentation by means of task migration. In this study, we studied many migration ways for heterogeneous system aiming to minimize such problems.

Thus, we assumed two types of migration: local m_l (Algorithm 8) and external m_e (Algorithm 9).

As m_e involves task transfer from one cluster to another, some strategies were proposed to avoid unnecessary migrations and, consequently, system overload. They are

1. checking all clusters that have available processors;
2. analyzing which jobs have their tasks at the beginning of the queue of idle processors;
3. based on the analysis above, checking a job which has the lowest number of tasks and that is less than or equal the number of idle processors;
4. finally, migrating job tasks which have the lowest number of tasks.

Algorithm 8 local migration(cluster)

Input: one of the grid clusters
Output: T (migration done) and F (migration did not happen)

```
1: set S ← machines available in the clust
2: set T ← empty
3: for i ← 1 to clust.number_of_machines() do
4:    if (clust.machines[i].queue[1]) and (clust.run(cluster.machines[i].queue[1]) =
      F) then
5:       T ← T U (clust.machines[i].queue[1])
6:    end if
7: end for
8: task ← T.shorter_number_migration()
9: if (number_of_migration ≤ S.cardinality()) then
10:   for i ← 1 to clust.number_of_machines() do
11:      if (clust.machine[i]_queue(task.id_job) = T)
         and (clust.machine[i].queue[1].id_job) ≠ (task.id_job) then
12:         clust.migration(clust.machine[i].task_with_id(task.id_job),
13:         S.shorter_queue())
14:      end if
15:   end for
16:   return T
17: end if
18: return F
```

Figure 8 presents a migration scenario. Processors P_1, P_2 and P_3 are available, tasks $v_{1,1}$ and $v_{2,1}$ are, respectively, at the beginning of processors queues P_1 and P_2, and task $v_{3,1}$ is in P_6 queue, the latter is busy and presenting other tasks in queue ahead of task $v_{3,1}$. Therefore, to ensure that tasks $v_{1,1}$, $v_{2,1}$ and $v_{3,1} \in J_1$ are immediately taken, $v_{3,1}$ is migrated to P_3.

Algorithm 9 external migration(cluster[n])

Input: all n clusters of the grid
Output: T (migration done) and F (migration did not happen)

```
1: set S ← empty
2: set T ← empty
3: for i ← 1 to n do
4:    for j ← 1 to cluster[i].number_machines() do
5:       if (exist_cluster[i].machine[j].queue[1])
         and (cluster.run(cluster[i].machine[j].queue[1]) = F) then
6:          S ← S U (cluster[i].machine[j].queue[1])
7:       end if
8:    end for
9: end for
10: job ← capture_job_id(T.shorter_number_task().id_job)
11: for i ← 1 to n do
12:    if (cluster[i].number_machine_available() ≥ job.number_task) then
13:       T ← T U (cluster[i])
14:    end if
15: end for
16: cluster_target ← T.minimum_machine_available
17: amount_task ← 0
18: for i ← 1 to cluster_target.number_of_machine() do
19:    if (not_exist_target.number_machine[i].running)
       and (amount_task ≤ job.number_task) then
20:       migrate(job, cluster_target.machine[i])
21:       amount_task ← amount_task + 1
22:    end if
23: end for
```

During task migration, destination processors are reserved in order to prevent that other tasks can use them. Reserving a destination processor will ensure that the migrated tasks start their executions immediately. It is important to note that the m_e is only applied when the m_l does not solve the problem.

Migration Strategies were applied in scheduling algorithms: AFCFS, LJFS and LXF (Section 5), which were used as a queue scheduler in the simulation environment. Therefore, these algorithms with migration will be defined as AFCFSm, LJFSm and LXFm. Scheduling hierarchy requires that, firstly, the scheduling algorithms are run (AFCFS, LXF or LJFS), and then, the m_l migration tries to dispatch the jobs not allocated by the scheduling algorithm. The m_e migration will only be used in an attempt of using more resources.

In the next section, it will be described the performance metrics that were applied to analyze the system model behavior in different situations.

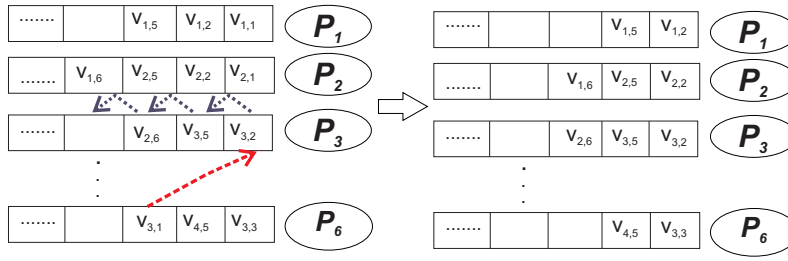


Figure 8. Example of a migration scenario

7 PERFORMANCE METRICS

In this study, the following performance metrics were applied: Average Waiting Time (AWT), Average Response Time (ART), Loss of Capacity (LoC) and Utilization (U) [38, 39, 40, 41], all in order to analyze the schedulers performance in different situations, as well as the system behavior in different contexts.

7.1 AWT

AWT measures the job average waiting time in the system, $AWT = \frac{1}{w} \times \sum_{j=1}^{w} wt(j)$ in which $wt(j)$ measures the time between the job arrival in the system and the beginning of its execution, and w is the total number of job executed.

7.2 ART

The metric response time (in seconds) measures the time interval between the job arrival in the system until the end of its execution, $ART = \frac{1}{w} \times \sum_{j=1}^{w} rt(j)$ in which $rt(j)$ represents the job response time and w is the total number of jobs executed.

7.3 LoC

This metric is relevant to measure both the use and the fragmentation of the system. Then, fragmentation happens when

1. there are tasks waiting in queue to execute;
2. there are idle nodes, but they still cannot perform tasks on hold.

LoC metric has been used in some studies, such as [26, 27, 38, 39, 40, 41]. In this work, LoC metric is calculated as follows:

$$LoC = \frac{\sum_{j=1}^{q-1} n_j(t_{j+1} - t_j)\delta_j}{N(t_q - t_1)} \times 100, \tag{2}$$

n_j represents the idle processors number during the time $(t_{j+1} - t_j)$; N is the total number of processors in the system; $t_q - t_1$ represents the arrival time of the first job in the system and the output of the latter one; and δ_j is the real condition of processor and jobs in the system. If $\delta_j = 1$, it indicates the existence of available processors to execute at least one job in queue by the moment a new job is dispatched; and $\delta_j = 0$ indicates that the queues are empty or that does not exist in queues jobs of size less than or equal to the number of idle processors.

Bellow is an example of LoC calculation, with a total of $N = 96$ processors, see Table 2.

$$LoC = \left[\frac{5(10-0)1 + 3(13-10)0 + 6(17-13)1 + 4(30-17)0 + 8(100-30)1}{96(100-0)}\right] \times 100 \doteq 6.6\,\% \quad (3)$$

t_j	δ_j and n_j
$t_1 = 0$; $t_2 = 10$	$\delta_1 = 1$ and $n_1 = 5$
$t_2 = 10$; $t_3 = 13$	$\delta_2 = 0$ and $n_2 = 3$
$t_3 = 13$; $t_4 = 17$	$\delta_3 = 1$ and $n_3 = 6$
$t_4 = 17$; $t_5 = 30$	$\delta_4 = 0$ and $n_4 = 4$
$t_5 = 30$; $t_q = 100$	$\delta_5 = 1$ and $n_5 = 8$

Table 2. LoC calculation example

The result corresponds to the fragmentation occurred in the system of the interval time $t_q - t_1 = 100$ (6.6 % of system fragmentation).

7.4 Utilization

In simulation studies, the utilization rate (U) of clusters is simply an indirect measure of *Makespan* [42, 45], calculation is given by Equation (4):

$$U = \frac{\sum_{j=1}^{w} s_j \times rt(j)}{Makespan \times N} \quad (4)$$

where U is the clustering utilization rate, s_j represents the task number of a job J_j and, consequently, as each job task must be performed in a separate processor at the same time, s_j also expresses the number of processors required to execute it, and *Makespan* is the difference between the initial execution time of the first job and the end time of the last one.

8 SIMULATION AND ANALYSIS OF RESULTS

8.1 Input Parameters

Simulations were carried out on GSMSim system, Java implemented, which was developed in the GrPeC Laboratory of UFC, allowing entity simulation in parallel

and distributed computing systems, such as users, applications, resource managers and schedulers.

The simulation environment used for the experiment consists of heterogeneous clusters with 128 and 256 processors, respectively, belonging to the same administrative domain. GD receives the necessary information from each cluster. Communication among processors is a free containment. What is more, latency (Section 3) is included in the job time service.

The simulator receives a workload as input and, according to scheduling policy of the current scheduling, makes a decision in order to meet the user demands. For environment analysis, many traces extracted from a real distributed environment were used [32, 33]. In addition, many tests were performed with heterogeneous and homogeneous environment using different workloads. However, in this article, the workload installed in the GSMSim is from the repository [32], which is provided by the HPC Systems of the San Diego Supercomputer Center group (SDSC). This SDSC load consists of 3 000 jobs totalling 140 441 tasks, which are described in tuple (id_j, at_j, s_j, pt_j) (Subsection 3.1). These jobs have very different small, medium, or large characteristics, such as: 1 860 jobs require 32 processors; 30, 90, 60, 60, 570 and 330 jobs require 1, 2, 4, 8, 64 and 128 processors, respectively. Therefore, on average, 366 tasks are handled by each processor.

For simulation, two scenarios were proposed: (i) In the first scenario (S1), it was used the OLB algorithm in LD, in order to assign tasks to queues in a random way to available processors; (ii) in the second scenario (S2), it was used the JSEQ algorithm in LD, in order to assign task to queues, according to tasks execution time on processors.

It is worth mentioning that the scheduling queues: AFCFS, LJFS and LXF (without migration) and AFCFSm, LJFSm and LXFm (with migration) were applied both in S1 and S2. For each scenario, ten simulations were executed and from that it was made the calculation of the average values of waiting times, response times, clustering percentage use and LoC. In each scheduling algorithm, previously mentioned, a 95 % confidence interval for average response time was used.

In the next section we present the results of simulations performed using the metrics described in Section 7. These results describe the impact on system performance mentioned above, regarding the migration applied in gang schedulers: AFCFS, LJFS and LXF. Furthermore, the impact of OLB (scenario S1) and JSEQ (scenario S2) in LD will be analyzed.

8.2 Average Response Time vs. Number of Executed Jobs

8.2.1 Scenario S1 – Using OLB Algorithm and Queue Schedulers

Figure 9 (scenario S1) shows the ART, using OLB algorithm and schedulers AFCFS, LXF and LJFS, and AFCFSm, LXFm and LJFSm, respectively, in which the x-axis represents the quantity of executed jobs. In this scenario, AFCFS algorithm submitted the lowest ART in all ranges of executed jobs regarding to LXF and

LJFS. LXF policy showed better results regarding to LJFS. This is justified because LXF policy tends to favor jobs that have higher XF (Subsection 5.3), differently from LJFS, which aims to benefit larger jobs, since it is not always true that the system offers processors to meet the smaller jobs causing an increase in response time.

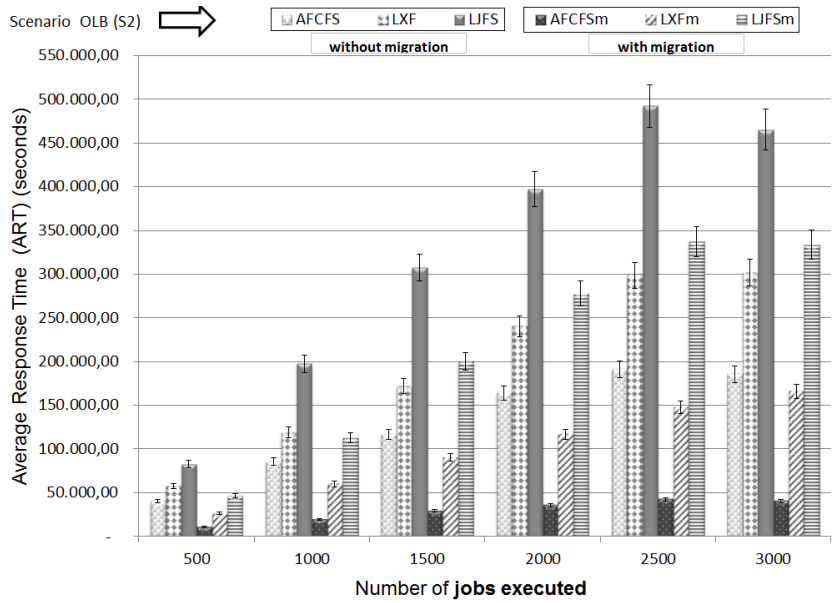


Figure 9. Scenario OLB – ART *versus* number of executed jobs

As shown in Figure 9 and Table 3, AFCFSm, LXFm and LJFSm algorithms show a significant decrease in ART concerning them without migration. This shows that using migration causes a big impact on response time. Therefore, the suggested method was able to use available processors more efficiently reducing the jobs response time. AFCFSm policy visibly presented the best result.

Number of Jobs	**Average Response Time (ART) – Seconds**					
	AFCFS	**AFCFSm**	**LXF**	**LXFm**	**LJFS**	**LJFSm**
500	40 133.44	10 719.64	57 826.43	26 209.00	82 580.55	46 161.78
1 000	85 287.65	19 498.69	118 982.20	59 897.42	197 405.05	113 076.48
1 500	116 247.61	29 175 47	171 619.96	90 376.76	307 561.28	200 460.79
2 000	164 252.48	35 410.80	240 306.39	116 669.94	397 360.80	277 986.15
2 500	190 661.94	42 315.17	298 628.58	147 463.46	492 325.32	337 005.26
3 000	185 395.03	40 170.85	301.549 44	165.735 51	465 193.56	333 426.95

Table 3. Scenario S1 – using the OLB algorithm and queue schedulers

8.2.2 Scenario S2 – Using JSEQ Algorithm and Queue Schedulers

In Figure 10 (scenario S2), it is shown the ART using JSEQ algorithm and schedulers AFCFS, LXF and LJFS, and AFCFSm, LXFm and LJFSm, respectively, in which x-axis represents the quantity of executed jobs. In this scenario, AFCFS algorithm had been presented the lowest ART in all quantities of jobs performed regarding to LXF and LJFS. LXF also showed better results regarding to LJFS. According to Subsection 8.2.1, LXF tends to favor jobs with higher XF.

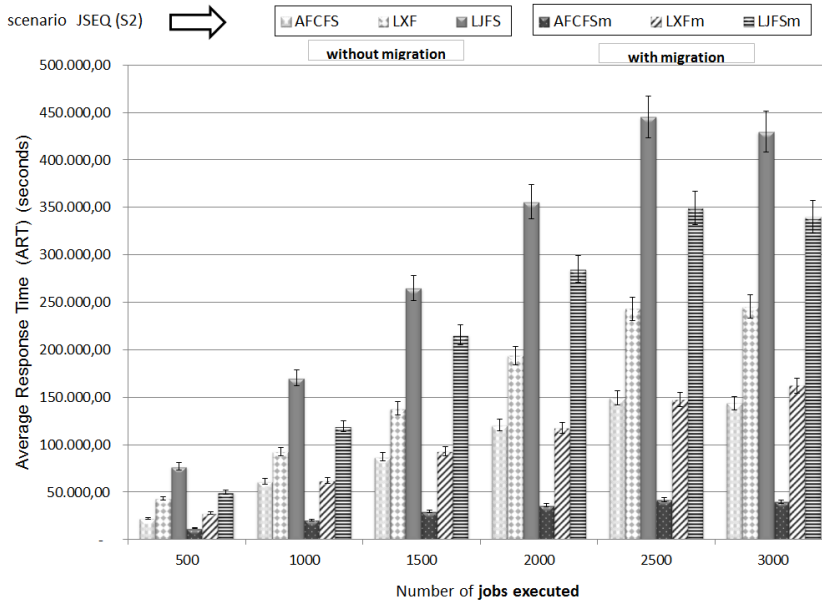


Figure 10. Scenario JSEQ – ART *versus* number of executed jobs

AFCFSm, LXFm and LJFSm algorithms in S2 (Table 4) have confirmed that the migration technique reduces response time, because it uses the available processors more efficiently.

Comparing results of AFCFS, LXF and LJFS algorithms in scenarios S1 and S2 (Tables 3 and 4), it is noted that in S2, ART is considerably reduced regardless of the scheduler queue used. This shows that JSEQ distributes tasks in queues more fairly. The information on total value of the task processing, i.e., the task processing time in queue plus the existence or not of the task that is running on processor, imply the task waiting time reduction and, consequently, the response time. On the other hand, algorithms with migration in scenario S1 (Figure 9) present ARTs similar to S2.

Based on the above, the results of AFCFSm, LXFm, and LJFSm in both scenarios are satisfactory, since they have reduced ART using the migration proposed mechanisms.

Number of Jobs	Average Response Time (ART) – Seconds					
	AFCFS	AFCFSm	LXF	LXFm	LJFS	LJFSm
500	22 432.98	11 940.57	43 339.01	27 840.66	76 982.39	50 044.96
1 000	61 298.36	20 463.54	92 728.93	62 541.50	170 497.66	119 391.59
1 500	87 077.07	29 937.91	138 066.91	93 214.22	265 064.16	215 580.81
2 000	120 630.94	36 003.33	193 861.07	117 629.26	355 929.28	284 934.41
2 500	149 538.01	41 998.61	242 923.46	147 423.36	445 306.99	349 568.18
3 000	143 846.61	40 030.39	245 386.95	162 278.49	429 514.51	340 029.61

Table 4. Scenario S2 – using the JSEQ algorithm and queue schedulers

8.3 Loss of Capacity in the System

8.3.1 Scenario S1 – Using OLB Algorithm and Queue Schedulers and Scenario S2 – Using JSEQ Algorithm and Queue Schedulers

In Figure 11 (scenario S1) and Figure 12 (scenario S2), the Loss of Capacity (in percentage) in the system is detailed. Comparing both graphic scenarios, the results of scheduling policies AFCFS, LXF and LJFS, and AFCFSm, LXFm and LJFSm showed equivalent LoC percentages. Using OLB or JSEQ implemented in LD does not influence the LoC metric results.

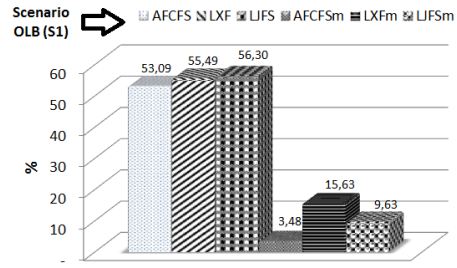


Figure 11. S1 – (%) LoC in the system

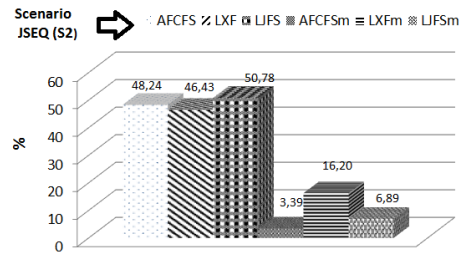


Figure 12. S2 – (%) LoC in the system

In scenarios S1 and S2, the AFCFS, LXF and LJFS algorithms cause approximately on average LoC = 50 % in the system. Analyzing AFCFSm, LXFm and LJFSm, respectively, AFCFSm presents percentages of LoC = 3.48 % and LoC = 3.39 %, lower results regarding to LXFm (LoC = 16.63 % and LoC = 16.2 %), and LJFSm (LoC = 9.63 % and LoC = 6.89 %). This implies that AFCFSm dispatches jobs more effectively, minimising system fragmentation. LXFm tends to present greater fragmentation concerning to LJFSm. The results confirm that scheduling algorithms with migration minimizes system fragmentation.

8.4 Cluster Utilization Rate

8.4.1 Scenario S1 – Using OLB Algorithm and Queue Schedulers and Scenario S2 – Using JSEQ Algorithm and Queue Schedulers

In Figure 13 (scenario S1) and Figure 14 (scenario S2), it is shown the percentage of cluster utilization regarding to interaction number. It is considered an interaction the job arrival to the GD and its completion. Comparing scenarios S1 and S2, AFCFS, LXF and LJFS and AFCFSm, LXFm and LJFSm algorithms present similar average use of resources.

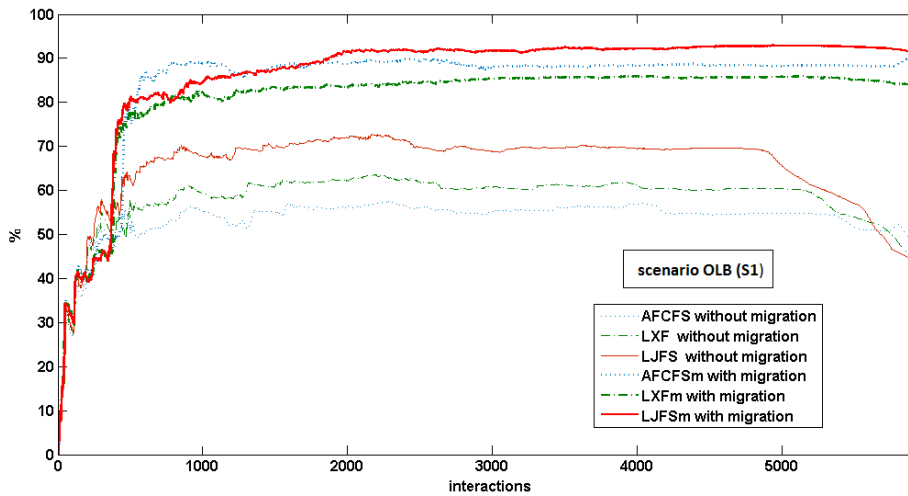


Figure 13. Scenario S1 – utilization rate (%)

In AFCFS, LXF and LJFS, in Figures 13 and 14, the clustering use average remains constant in intervals of 1 000–4 900. AFCFS presents lower percentages regarding to LXF and LJFS. This is because the algorithm tends to favor smaller jobs, generating an increase in idle processors. But LXF and LJFS have better results in the clusters use. Analyzing AFCFSm, LXFm and LJFSm, Figures 13 and 14, confirmed that algorithms with migration strategy are more efficient when

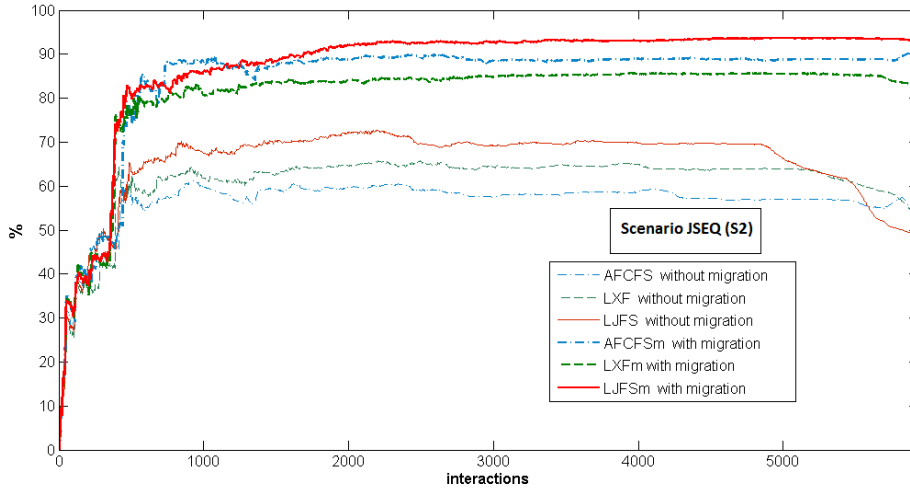


Figure 14. Scenario S2 – utilization rate (%)

it comes to the use of resources. In addition, we can see in Figures 13 and 14 that after 5 000 interactions a considerable decline happens; this implies that all jobs have been serviced and the processors are becoming available.

9 CONCLUSIONS AND FUTURE WORK

This study has proposed and implemented a Multicore Multicluster GSMSim simulation system, using a two-layer hierarchical structure, GD and LD. GSMSim was developed in order to analyze the scheduling performance in different situations, as well as the environment behavior in different contexts. In LD, two scenarios were implemented and adapted, OLB (S1) and JSEQ (S2), in order to distribute tasks efficiently in the system, which act before scheduling queues. Additionally, the schedulers AFCFS, LXF and LJFS were used and adapted to gang technique for scheduling tasks in processor queues. As aforementioned, such schedulers cause environment fragmentation, therefore, migration mechanisms were implemented in order to minimize this problem. In the experiment analysis, performance metrics were used aiming to assess the scheduler behavior in different situations.

In scenario S2, the tasks were distributed more efficiently in queues, minimizing the task waiting time. As a consequence, the scheduling queues AFCFS, LXF and LJFS showed the most significant results regarding to ART metric in scenario S1. In these scenarios, AFCFSm, LXFm and LJFSm presented satisfactory ARTs. This implies that the suggested migration technique was able to use idle processors more efficiently, thereby reducing system fragmentation. Adapted by nodes in this context, the metric LoC measures the impact that schedulers cause in the system, regarding to fragmentation. Results obtained (Figures 11 and 12), in AFCFS, LXF

and LJFS cause ≈ 50 % of system fragmentation. In the proposed mechanisms, the fragmentation was considerably reduced in AFCFSm (3.48 % and 3.39 %), LXFm (15.63 % and 16.2 %) and LJFSm (9.63 % and 6.89 %) (Figures 11 and 12). Regarding to clustering metric (Figures 13 and 14), it was confirmed that migration technique reduces the number of idle processors in the system, as well as fragmentation.

The results showed that there was a fragmentation reduction using task migration among processor queues in a heterogeneous multicluster environment, as well as a better use of them, implying operating cost reduction on the part of providers, meeting the expectations of users QoS. It is worth pointing out that scenario S2 presented satisfactory results in all metrics, unlike S1, which in ART metric (AFCFS, LXF and LJFS) was not as efficient. AFCFSm presented the best results in both scenarios.

As future work, a wide research may be carried out in the scheduling field for computational grids. In this study, only OLB and JSEQ algorithms were used in LD. Then, we intended to apply other heuristics in order to analyze the system behavior in different approaches. Besides, a new scheduling heuristics for applications such as DAG could be created. In another perspective, the migration techniques applied in AFCFS, LXF and LJFS could be implemented in other schedulers, e.g., in genetic algorithms, comparing them with the ones used in this work. Moreover, a proposed model validation in real context was evaluating a large number of experimental results.

Acknowledgment

We thank the CAPES (Coordination for the Improvement of Higher Education Personnel) for the financial support, the National Postdoctoral Program Department of Computer Science – UERN/UFERSA, SEDUC-CE and IFRN.

REFERENCES

[1] Luo, H.—Mu, D.—Deng, Z.—Wang, X.: A Review of Job Scheduling for Grid Computing. Application Research of Computer, Vol. 5, 2005, pp. 16–19 (in Chinese).

[2] Cook, S. A.: The Complexity of Theorem-Proving Procedures. Proceedings of the 3rd Annual ACM Symposium on Theory of Computing (STOC '71), 1971, pp. 151–158, doi: 10.1145/800157.805047.

[3] Ullman, J. D.: NP-Complete Scheduling Problems. Journal of Computer and System Sciences, Vol. 10, 1975, pp. 384–393, doi: 10.1016/S0022-0000(75)80008-0.

[4] Topcuoglu, H.—Hariri, S.—Wu, M.-Y: Performance-Effective and Low Complexity Task Scheduling for Heterogeneous Computing. IEEE Transactions on Parallel and Distributed Systems, Vol. 13, 2002, No. 3, pp. 260–274, doi: 10.1109/71.993206.

[5] Feitelson, D. G.: Packing Schemes for Gang Scheduling. In: Feitelson, D. G., Rudolph, L. (Eds.): Job Scheduling Strategies for Parallel Processing (JSSPP 1996).

Springer, Berlin, Heidelberg, Lecture Notes in Computer Science, Vol. 1162, 1996, pp. 89–110, doi: 10.1007/BFb0022289.

[6] Casavant, T. L.—Kuhl, J. G.: A Taxonomy of Scheduling in General-Purpose Distributed Computing Systems. IEEE Transactions on Software Engineering, Vol. 14, 1988, No. 2, pp. 141–154, doi: 10.1109/32.4634.

[7] Wieczorek, M.—Hoheisel, A.—Prodan, R.: Towards a General Model of the Multi-Criteria Workflow Scheduling on the Grid. Future Generation Computer Systems, Vol. 25, 2009, No. 3, pp. 237–256, doi: 10.1016/j.future.2008.09.002.

[8] Smanchat, S.—Viriyapant, K.: Taxonomies of Workflow Scheduling Problem and Techniques in the Cloud. Future Generation Computer Systems, Vol. 52, 2015, pp. 1–12, doi: 10.1016/j.future.2015.04.019.

[9] Lopes, R. V.—Menascé, D.: A Taxonomy of Job Scheduling on Distributed Computing Systems. IEEE Transactions on Parallel and Distributed Systems, Vol. 27, 2016, No. 12, pp. 3412–3428, doi: 10.1109/TPDS.2016.2537821.

[10] Maheswaran, M.—Ali, S.—Siegal, H. J.—Hensgen, D.—Freund, R. F.: Dynamic Matching and Scheduling of a Class of Independent Tasks Onto Heterogeneous Computing Systems. Proceedings of the Eighth Heterogeneous Computing Workshop (HCW '99), 1999, pp. 30–44, doi: 10.1109/HCW.1999.765094.

[11] Braun, T. D.—Siegel, H. J.—Beck, N.—Bölöni, L. L.—Maheswaran, M.—Reuther, A. I.—Robertson, J. P.—Theys, M. D.—Yao, B.—Hensgen, D.—Freund, R. F.: A Comparison of Eleven Static Heuristics for Mapping a Class of Independent Tasks onto Heterogeneous Distributed Computing Systems. Journal of Parallel and Distributed Computing, Vol. 61, 2001, No. 6, pp. 870–837, doi: 10.1006/jpdc.2000.1714.

[12] Wang, J.—Abu-Ghazaleh, N.—Ponomarev, D.: Controlled Contention: Balancing Contention and Reservation in Multicore Application Scheduling. 2015 IEEE International Parallel and Distributed Processing Symposium (IPDPS), IEEE, 2015, doi: 10.1109/IPDPS.2015.62.

[13] Feitelson, D. G.: Resampling with Feedback – A New Paradigm of Using Workload Data for Performance Evaluation. In: Dutot, P. F., Trystram, D. (Eds.): Euro-Par 2016: Parallel Processing. Springer, Cham, Lecture Notes in Computer Science, Vol. 9833, 2016, pp. 3–21, doi: 10.1007/978-3-319-43659-3_1.

[14] Ousterhout, J.: Scheduling Techniques for Concurrent Systems. Proceedings of the 3rd International Conference on Distributed Computing Systems, 1982, pp. 22–30.

[15] Zhou, B. B.—Mackerras, P.—Johnson, C. W.—Walsh, D.—Brent, R. P.: An Efficient Resource Allocation Scheme for Gang Scheduling. IEEE Computer Society 1st International Workshop on Cluster Computing (IWCC '99), 1999, doi: 10.1109/IWCC.1999.810824.

[16] González, J. C.: Coordinated Scheduling and Dynamic Performance Analysis in Multiprocessors Systems. Ph.D. Thesis, Universitat Politècnica de Catalunya, Departament d'Arquitectura de Computadors, 2002.

[17] Feitelson, D. G.—Rudolph, L.—Schwiegelshohn, U.: Parallel Job Scheduling – A Status Report. In: Feitelson, D. G., Rudolph, L., Schwiegelshohn, U. (Eds.): Job Scheduling Strategies for Parallel Processing (JSSPP 2004). Springer, Berlin,

Heidelberg, Lecture Notes in Computer Science, Vol. 3277, 2004, pp. 1–16, doi: 10.1007/11407522_1.

[18] Karatza, H. D.: Performance of Gang Scheduling Strategies in a Parallel System. Simulation Modelling Practice and Theory, Vol. 17, 2009, No. 2, pp. 430–441, doi: 10.1016/j.simpat.2008.10.001.

[19] Moschakis, I. A.—Karatza, H. D.: Evaluation of Gang Scheduling Performance and Cost in a Cloud Computing System. The Journal of Supercomputing, Vol. 59, 2012, No. 2, pp. 975–992, doi: 10.1007/s11227-010-0481-4.

[20] Hao, Y.—Wang, L.—Zheng, M.: An Adaptive Algorithm for Scheduling Parallel Jobs in Meteorological Cloud. Knowledge-Based Systems, Vol. 98, 2016, pp. 226–240, doi: 10.1016/j.knosys.2016.01.038.

[21] Tomás, L.—Caminero, B.—Carrión, C.: Improving Grid Resource Usage: Metrics for Measuring Fragmentation. 2012 12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing, 2012, pp. 352–359, doi: 10.1109/CCGrid.2012.63.

[22] Feitelson, D. G.: A Survey of Scheduling in Multiprogrammed Parallel Systems. Research Report, IBM T. J. Watson Research Center, 1994.

[23] Manickam, V.—Aravind, A.: A Fair and Efficient Gang Scheduling Algorithm for Multicore Processors. In: Venugopal, K. R., Patnaik, L. M. (Eds.): Wireless Networks and Computational Intelligence (ICIP 2012). Springer, Berlin, Heidelberg, Communications in Computer and Information Science, Vol. 292, 2012, pp. 467–476, doi: 10.1007/978-3-642-31686-9_54.

[24] Liu, X.—Chen, B.—Qiu, X.—Cai, Y.—Huang, K.: Scheduling Parallel Jobs Using Migration and Consolidation in the Cloud. Mathematical Problems in Engineering, Hindawi Publishing Corporation, Vol. 2012, Art. No. 695757, 18 pp., doi: 10.1155/2012/695757.

[25] Papazachos, Z. C.—Karatza, D. H.: Gang Scheduling in Multi-Core Clusters Implementing Migrations. Future Generation Computer Systems, Vol. 27, 2011, No. 8, pp. 1153–1165, doi: 10.1016/j.future.2011.02.010.

[26] Pinto, F. A. P.—Chaves, C. B.—de M. Leite, L. G.—Vasconcelos, F. H. L.—Barroso, G. C.: Analysis of Scheduling Algorithms with Migration Strategies in Distributed Systems. The Tenth International Conference on Networking and Services (ICNS 2014), 2014, pp. 12–17.

[27] Pinto, F. A. P.—Leite de Moura, L. G.—Barroso, G. C.—Aguilar, M. M. F.: Algorithms Scheduling with Migration Strategies for Reducing Fragmentation in Distributed Systems. IEEE Latin America Transactions, Vol. 13, 2015, No. 3, pp. 762–768, doi: 10.1109/TLA.2015.7069102.

[28] Marsan, M. A.—Balbo, G.—Conte, G.: Performance Models of Multiprocessor Systems. MIT Press, 280 pp., 1987.

[29] Garg, S. K.—Venugopal, S.—Broberg, J.—Buyya, R.: Double Auction-Inspired Meta-Scheduling of Parallel Applications on Global Grids. Journal of Parallel and Distributed Computing, Vol. 73, 2012, No. 4, pp. 450–464, doi: 10.1016/j.jpdc.2012.09.012.

[30] Hockney, R. W.: The Communication Challenge for MPP: Intel Paragon and Meiko CS-2. Parallel Computing, Vol. 20, 1994, pp. 389–398, doi: 10.1016/S0167-8191(06)80021-9.

[31] Casanova, H.—Giersch, A.—Legrand, A.—Quinson, M.—Suter, F.: Versatile, Scalable, and Accurate Simulation of Distributed Applications and Platforms. Journal of Parallel and Distributed Computing, Elsevier, Vol. 74, 2014, No. 10, pp. 2899–2917, doi: 10.1016/j.jpdc.2014.06.008.

[32] Feitelson, D. G.: Job Scheduling in Multiprogrammed Parallel Systems. IBM Research Report RC 19790 (87657), 1997.

[33] The Standard Workload Format. Available at: `http://www.cs.huji.ac.il/labs/parallel/workload/`.

[34] Lin, H.-C.—Raghavendra, C. S.: An Analysis of the Join the Shortest Queue (JSQ) Policy. Proceedings of the 12th International Conference on Distributed Computing Systems, 1992, doi: 10.1109/ICDCS.1992.235020.

[35] Papazachos, Z. C.—Karatza, H. D.: Performance Evaluation of Bag of Gangs Scheduling in a Heterogeneous Distributed System. Journal of Systems and Software, Vol. 83, 2010, No. 8, pp. 1346–1354, doi: 10.1016/j.jss.2010.01.009.

[36] Vasupongayya, S.—Prasitsupparote, A.: Extending Goal-Oriented Parallel Computer Job Scheduling Policies to Heterogeneous Systems. The Journal of Supercomputing, Vol. 65, 2013, No. 3, pp. 1223–1242, doi: 10.1007/s11227-013-0879-x.

[37] Sedgewick, R.—Wayne, K.: Algorithms. 4th edition. Addison-Wesley, 2011.

[38] Zhang, Y.—Franke, H.—Moreira, J. E.—Sivasubramaniam, A.: The Impact of Migration on Parallel Job Scheduling for Distributed Systems. In: Bode, A., Ludwig, T., Karl, W., Wismüller, R. (Eds.): Euro-Par 2000 Parallel Processing. Springer, Berlin, Heidelberg, Lecture Notes in Computer Science, Vol. 1900, 2000, pp. 242–251, doi: 10.1007/3-540-44520-X_33.

[39] Leung, V. J.—Sabin, G.—Sadayappan, P.: Parallel Job Scheduling Policies to Improve Fairness: A Case Study. 2010 39th International Conference on Parallel Processing Workshops, 2010, pp. 346–353, doi: 10.1109/ICPPW.2010.48.

[40] Tang, W.—Lan, Z.—Desai, N.—Buettner, D.—Yu, Y.: Reducing Fragmentation on Torus-Connected Supercomputers. 2011 IEEE International Parallel and Distributed Processing Symposium (IPDPS), 2011, doi: 10.1109/IPDPS.2011.82.

[41] Tang, W.—Ren, D.—Lan, Z.—Desai, N.: Adaptive Metric-Aware Job Scheduling for Production Supercomputers. 2012 41st International Conference on Parallel Processing Workshops, IEEE, 2012, doi: 10.1109/ICPPW.2012.17.

[42] Burkimsher, A.—Bate, I.—Indrusiak, L. S.: A Survey of Scheduling Metrics and an Improved Ordering Policy for List Schedulers Operating on Workloads with Dependencies and a Wide Variation in Execution Times. Future Generation Computer Systems, 2012, Vol. 29, 2013, No. 8, pp. 2009–2025, doi: 10.1016/j.future.2012.12.005.

[43] Hamid, N.—Walters, R. J.—Wills, G. B.: An Architecture for Measuring Network Performance in Multi-Core Multi-Cluster Architecture (MCMCA). International Journal of Computer Theory and Engineering, Vol. 7, 2015, pp. 57–61, doi: 10.7763/IJCTE.2015.V7.930.

[44] Salehi, M. A.—Smith, J.—Maciejewski, A. A.—Siegel, H. J.—Chong, E. K. P.—Apodaca, J.—Briceño, L. D.—Renner, T.—Shestak, V.—Ladd, J.—Sutton, A.—Janovy, D.—Govindasamy, S.—Alqudah, A.—Dewri, R.—Prakash, P.: Stochastic-Based Robust Dynamic Resource Allocation for Independent Tasks in a Heterogeneous Computing System. Journal of Parallel and Distributed Computing, Vol. 97, 2016, pp. 96–111, doi: 10.1016/j.jpdc.2016.06.008.

[45] Xie, G.—Zeng, G.—Liu, L.—Li, R.—Li, K.: Mixed Real-Time Scheduling of Multiple DAGs-Based Applications on Heterogeneous Multi-Core Processors. Microprocessors and Microsystems, Vol. 47, 2016, Part A, pp. 93–103, doi: 10.1016/j.micpro.2016.04.007.

[46] Dehlaghi-Ghadim, A.—Entezari-Maleki, R.—Movaghar, A.: Cost-Efficient Sheduling for Deadline Constrained Grid Workflows. Computing and Informatics, Vol. 37, 2018, No. 4, pp. 838–864, doi: 10.4149/cai_2018_4_838.

[47] Langer, T.—Osinski, L.—Mottok, J.: A Survey of Parallel Hard-Real Time Scheduling on Task Models and Scheduling Approaches. 30$^{\text{th}}$ International Conference on Architecture of Computing Systems (ARCS 2017), 2017.

[48] Kang, W.—Kim, J.: Effective Scheduling of Grid Resources Using Failure Prediction. Computing and Informatics, Vol. 35, 2016, No. 2, pp. 369–390.

[49] Zakarya, M.—Gillam, L.: Energy Efficient Computing, Clusters, Grids and Clouds: A Taxonomy and Survey. Sustainable Computing: Informatics and Systems, Vol. 14, 2017, pp. 13–33, doi: 10.1016/j.suscom.2017.03.002.

Francisca A. P. Pinto received her Postdoctoral degree from the Department of Computer Science at the Universidade Estadual do Rio Grande do Norte (State University of Rio Grande do Norte) (UERN) in 2018. She is Coordinator of the Research Group on Applied Computational Modeling. She graduated in mathematics and received her Master degree and Ph.D. in teleinformatics engineering from the Universidade Federal do Ceará (Federal University of Ceará) (UFC). Her research interests include mathematics and computer science, distributed systems, sheduling algorithms, Petri nets, distance education and semantic web.

Henrique J. A. Holanda holds his degree in computer science from the UFC and the postdoctoral degree in software engineering from École Polytechnique de Montréal. He is currently Associate Professor IV at the UERN. His research interests include computer science with emphasis on software engineering, working mainly in the following subjects: anti-pattern, system specialist, software quality, Petri nets and human machine interface.

Carla K. de M. Marques holds her degree in computer science from the UFC and the Ph.D. in teleinformatics engineering from the UFC. She is currently Associate Professor IV and Permanent Professor of the postgraduate program in computer science at the UERN. Her research interests include computer science with emphasis on teleinformatics, working mainly on the following topics: dynamic reconfiguration, Petri nets, web servers, quality of service and architecture in grids.

Giovanni C. Barroso holds his Ph.D. in electrical engineering from the Universidade Federal da Paraíba (Federal University of Paraiba). He is Permanent Professor of the postgraduate program in teleinformatics engineering at the UFC. His research interests include electrical engineering and distance education, supervisory control, distributed systems and modeling in hybrid systems.

Computing and Informatics, Vol. 38, 2019, 421–453, doi: 10.31577/cai_2019_2_421

RDGC: A REUSE DISTANCE-BASED APPROACH TO GPU CACHE PERFORMANCE ANALYSIS

Mohsen Kiani, Amir Rajabzadeh

Department of Computer Engineering and Information Technology
Engineering Faculty, Razi University
Taghe-Bostan, Kermanshah, Iran
e-mail: {kiani.mohsen, rajabzadeh}@razi.ac.ir

Abstract. In the present paper, we propose RDGC, a reuse distance-based performance analysis approach for GPU cache hierarchy. RDGC models the thread-level parallelism in GPUs to generate appropriate cache reference sequence. Further, reuse distance analysis is extended to model the multi-partition/multi-port parallel caches and employed by RDGC to analyze GPU cache memories. RDGC can be utilized for architectural space exploration and parallel application development through providing hit ratios and transaction counts. The results of the present study demonstrate that the proposed model has an average error of 3.72 % and 4.5 % (for L1 and L2 hit ratios, respectively). The results also indicate that the slowdown of RDGC is equal to 47 000 times compared to hardware execution, while it is 59 times faster than GPGPU-Sim simulator.

Keywords: GPU cache memory, reuse distance analysis, performance modeling, hit ratio

Mathematics Subject Classification 2010: 68M20

1 INTRODUCTION

Many modern high performance computing systems rely on GPUs along with CPUs to deliver high amounts of computing power. Since GPU usage is no longer limited to the graphical processing applications, architectures of modern GPUs are modified towards the benefit of general computations. One of the most significant changes in GPU architectures is the utilization of cache memories in GPUs [1]. Cache memories

can alleviate the traditional problem of memory wall through exploiting the data localities which inherently exist in many general applications. Modern GPUs employ two levels of hardware-managed cache memories. Although cache hit ratios in GPUs are not generally as high as CPUs, the overall GPU performance is highly affected by cache performance in many data parallel applications [2]. In modern CPUs, approximately one-third of the chip area is devoted to cache memories, while the per-core cache size in GPUs is very limited. Moreover, since GPUs use thread switching, a huge number of in-flight threads run concurrently, what results in many attempts to access cache memory lines and causes cache thrashing. Hence, given the limited size of the available cache memory in GPUs, the detailed cache performance modeling is essential. For instance, hardware architects who intend to organize cache memories and application developers who work toward optimum application implementation would highly benefit from detailed cache performance modeling.

There are three main approaches to evaluating how processors function: measurement, simulation, and mathematical performance modeling [3]. To evaluate GPU cache memory performance, appropriate tools and techniques should be developed based on the architectural characteristics of GPUs. It should be noted that the existing CPU cache performance modeling techniques are not applicable for GPUs and require considerable modifications prior to use because of the substantial architectural differences between CPUs and GPUs.

In the present paper, a reuse distance-based approach, called RDGC, is proposed to analyze the performance of GPU cache memory hierarchy. RDGC embodies two models: logical and physical. In the former, the trace information is first extracted, then compressed, and finally ordered logically. In this case, the trace memory accesses ordering is performed regardless of the GPU physical resource limitation, i.e., for unlimited number of processing resources. Hence, the logical model is GPU independent. In the latter case, the physical limitations of a specific GPU, which are essential to the performance estimation of a given GPU, are modeled to define the cache reference sequence. The extended reuse distance (RD) analysis algorithm proposed by the present study is then applied by the physical model to generate the performance metrics for the cache reference sequence. The merit of using these two separate models is that the logical model is not specific to any GPU generation, and its outputs can be used for any GPU machines modeled by the physical model.

Given that GPUs place emphasis on parallelism and the fact that GPU caches may have multiple banks with multiple access ports, RD analysis algorithm [4] was extended in the present study to model such cache memories. In addition, since GPUs employ two levels of cache memories, two cache levels are modeled by RDGC: per-SM (Streaming Multiprocessor) private L1 caches and shared L2 cache.

RDGC provides hardware architects with exploration of GPU cache design space. Additionally, the presented method can be used by application developers to optimize the data locality exploited by cache memories. To analyze the performance of GPU cache memories, different cache design parameters were modeled, including capacity, associativity, block size, bypassing, mapping (indexing) function, and replacement policy. Further, several mapping policies of thread blocks to SMs

were modeled. Finally, the effects of L2 parallelisms were investigated. Moreover, the RDGC was validated against the performance counters provided by NVIDIA's NVPROF profiler. The Polybench/GPU applications [5] and several applications selected from Rodinia benchmarks [6] were executed on a Maxwell and a Kepler GPU, and the results provided by NVPROF were used to validate RDGC. Further, the performance of a selection of applications were evaluated for different cache memory parameters and GPU thread mapping policies.

RD analysis has been adapted for GPU cache memories in studies conducted by Tang et al. [7] and Nugteren et al. [8], in which only a single cache level (L1) is modeled. In this paper, the RD model presented by [8] was extended to include cache parallelism, i.e. multi-port and multi-bank cache memories. In addition, compared to previous studies, the present work is more comprehensive, and two levels of cache memories with different cache parameters are analyzed. Instead of solely generating hit ratios, the transaction counts were also provided by the model as a performance metric which is essential when modeling average memory access time [9]. Furthermore, RDGC was validated for newer GPU generations.

The utilization of cycle-accurate simulators for architectural space exploration is immensely popular with hardware architects. However, simulators are extremely slow and it is exceedingly time consuming to investigate the performance of different cache configurations using a cycle-accurate simulator. The slowdown of GPGPU-sim (V 3.2.2), as a popular simulator, is around 2760000 times for the workload in the present study. Applications with run-times of several milliseconds took hours to be simulated. RDGC has an average simulation slowdown of 47K times, thereby generating the demanded results within several minutes. In addition, the RDGC computations have a degree of parallelism and can be accelerated by parallel programming [10], whereas parallelizing simulators is challenging [11]. In addition, since we use RD analysis, the results of one simulation can be used to predict other cache organizations [12, 9], or it can be employed as a basis to estimate the total processor performance and power [13]. Consequently, RDGC can be used to narrow down the broad architectural space of cache organizations, and the optimal architecture candidates can be later simulated in more details. Last but not least, the older GPU generations are usually simulated by GPU simulators, but their pace of evolution is not in line with GPUs. For instance, the NVIDIA Fermi GPUs are simulated by GPGPU-Sim, while RDGC is designed based on the newer Maxwell GPU generation.

The main contributions of this paper are summarized as follows:

1. A reuse distance analysis algorithm is proposed for modeling the multi-port and multi-bank cache memories. Further, the effects of Miss Status Holding Registers (MSHRs) and cache memory latency are included in the presented model.
2. An appropriate model is developed to model the GPU thread level parallelism and generate the cache reference sequence.
3. Different cache organizational parameters are analyzed in this paper.

4. The effects of L2 cache parallelism in terms of multiple cache partitions and banks are analyzed to quantify the effects of parallelism on the resultant reuse profile.

The present paper embodies the following sections: Section 2 deals with literature review, and a background on NVIDIA GPUs and RD analysis is presented in Section 3. Next, RDGC is explained in Section 4. Later, the evaluation results are presented and discussed in Section 5, and finally, the paper is concluded in Section 6.

2 RELATED WORK

A great deal of studies have been conducted about cache performance modeling in CPUs, whereas the very same subject has not been dealt extensively in the case of GPUs. In the following, we review the related researches to the context of our study.

2.1 GPU Cycle-Accurate Simulators

Hardware architects rely on cycle-accurate simulators to explore the architectural space of GPUs, but its main limitation is the extreme slowdowns of simulators. Although detailed results are provided by simulators, they may fail to provide good insights into some detailed results about the architectural aspects of processors. Further, architectural space exploration with cycle-accurate simulation is very time consuming since it requires to simulate every one of architecture candidates. Some of the prime examples of GPU simulators are GPGPU-sim [14] and Multi2Sim [15].

2.2 Analytical and Empirical Performance Models

An analytical performance model was introduced for GPUs by Hong and Kim [16], but its main problem was that the cache memory effects were not addressed in the proposed model. Later, the said model was extended by Sim and Kim [17] to include the effects of cache memories, which were supposed to be known in advance. In another study performed by Baghsorkhi et al. [18], a hierarchical memory model, based on statistical sampling and trace file analysis, was proposed to predict the performance of the GPU's memory hierarchy. To generate an appropriate memory access sequence, the Monte Carlo simulation method was exploited by the authors of the said study. In another study performed by Huang et al., known as GPUMech [19], the interval analysis technique was extended, in which the parameters affecting the performance of GPUs were modeled, including the effects of multithreading, MSHRs limitation, and DRAM bandwidths. To determine the sequence of memory accesses, the Round Robin (RR) and Greedy-Then-Oldest (GTO) policies were employed. GPURoofline [20] is an empirical approach for performance evaluation and optimization of GPU applications towards observing the performance bottlenecks of applications and manually optimizing the performance of applications. Machine learning was adopted by some researchers to develop predicting performance models

for GPUs. For example, Dao et al. [21] concluded that linear analytical models fail to capture the effects of GPU memory systems and presented a machine learning-based model for GPUs that run the OpenCL kernels to accurately estimate the performance of running kernels. Recently, Kiani and Rajabzadeh proposed a model to approximate the locality in CUDA kernels with regular access patterns [22].

2.3 Reuse Distance-Based Cache Performance Modeling

Multicore CPUs: Although RD analysis is basically designed for single thread analysis, the prevalence of multicore CPUs has motivated many researches toward employing RD analysis for multicore CPUs. Both private and shared caches may exist across a multicore cache hierarchy, each requiring proper mechanisms to calculate the reuse profile. Ding and Chimbili [23] proposed a locality estimation model for multi-threaded applications. The authors modeled thread interleaving and data sharing to profile the locality in shared caches. Similarly, Jiang et al. [24] extended RD profile for shared caches by introducing Concurrent Reuse Distance (CRD) profiles. Their work relies on probabilistic models to estimate CRD profiles from the individual threads memory references. As the authors pointed out, in contrary to RD profiles, CRD profiles are not architecture independent. However, in many applications with similar memory behaviors across threads, CRD can be considered as a virtually hardware independent metric and once acquired for a given architecture, it can be estimated for other architectures [12, 9]. Schuff et al. [25] consider both private and shared caches in multicore systems and extended RD analysis to account for write-invalidation in private and inter-core data sharing in shared caches. Moreover, Wu and Yeung [26] consider loop-level parallelism in which the threads exhibit very similar memory behaviors. The authors used CRD profiles to predict reuse profiles for different core counts in Large-scale Chip Multi-Processors (LCMPs). Their method is useful to conduct core count and problem size scaling analysis. Later Wu and Yeung [12, 9] extended their prior work by employing Private RD (PRD) along with CRD profiles to explore the cache hierarchy architectural space in multicore CPUs. They show that using RD profiles the average memory access time, which is one of the most important performance parameters in CPUs, can be estimated using simple analytical models. Recently, Badamo et al. [13] employed RD analysis and analytical modeling to predict the performance and power consumption and identify power-efficient cache organizations in LCMPs.

Acquiring RD profiles for every possible cache organization is costly thus some techniques have been developed to reduce the analysis time. The first technique is prediction through which the RD profile is acquired for several hardware configurations and then predicted for all other configurations, hence extensively reduces the analysis cost [26]. Another alternative is using statistical sampling methods. In this technique, a small yet representative subset of the

memory references is analyzed which yields a similar profile achievable through full analysis [27]. In addition, RD analysis can be accelerated through parallel execution [10]. It should be noted that applying prediction, sampling, and parallelization techniques is not straightforward in the case of cycle-accurate simulation [11].

GPUs: GPU threads execute the same code (Single Instruction Multiple Threads), thus threads generally exhibit similar memory behaviors. Further, no coherency protocol is employed at L1 level, and only one shared L2 exists in GPUs (see Section 3). Consequently, when adapting RD analysis for GPUs, there is no need to model coherency effects as modeled in multicore CPUs.

Some researchers adapted RD analysis for GPU kernels. In [8], RD analysis algorithm was extended for GPUs to evaluate the performance of L1 cache memory. In addition, the trace file was generated by Ocelot and the access sequence was defined based on the RR scheduling policy. The results demonstrated that hit ratios were chiefly governed by cache capacity, associativity, and block size. However, they do not consider cache level parallelism, and, as the result of the present study shows, cache parallelism can significantly change the achieved reuse distance values. In addition, the authors do not model write accesses and in this article we include writes by considering write-evict policy (which is the policy used in GPUs). Further, Tang et al. also proposed the reuse distance-based algorithm for L1 cache analysis [7]. The problem was divided into two parts. Firstly, a stack (reuse) distance algorithm was developed for a single CUDA block in which the RR policy was assumed for warp scheduling. Secondly, the contention effects, caused by the simultaneous execution of multiple blocks on the very same SM, were modeled. Tang et al., however, do not give any detail regarding the way they modeled the GPU physical limitations. Moreover, they do not model the effects of MSHRs. Recently, RD analysis was employed by Wang et al. to analyze the access patterns of GPU applications [28]. They provide reuse distance breakdown calculated from the memory access information generated by GPGPU-sim. Since their approach relies on GPGPU-sim to generate the memory access information, a considerable time should be devoted for memory trace extraction, thus it is not a time-efficient approach. All in all, RD analysis in the context of GPUs is in its early stages and as a step forward, we try to enhance the existing algorithms by including both cache levels, cache parallelism, and modeling write accesses.

2.4 GPU Cache Memory Organizational Space Exploration

One of the main objectives of the present study is the analysis of the behaviors of cache memories in GPUs in the case of different cache organizations. The organizational space of cache memories and the architectural techniques for cache memories have been investigated in many previous studies [29]. Warp scheduling [30], cache prefetching [31], cache bypassing [32] and cache indexing [33] are among the most

important areas in cache memory organizational investigation that have received a great deal of attention.

3 FUNDAMENTALS: NVIDIA-GPU, CUDA, AND RD ANALYSIS

NVIDIA GPUs: NVIDIA GPUs consists of several streaming multiprocessors (SMs), memory controllers, and an L2 cache memory connected to an off-chip global memory shared between the SMs via an interconnection network. Each SM is composed of processing and memory resources. The former includes processing cores, load/store units, special function units (SFUs), and the latter includes a register file, a shared memory, and an L1 cache. The internal organization of SMs and memory hierarchy varies from one GPU generation to another.

Compute Unified Device Architecture (CUDA): This programming model was developed by NVIDIA for its GPUs towards the development of scalable GPU applications [34]. A CUDA application can be performed on different generations of CUDA-enabled GPUs, possibly with different number of computing resources. In the CUDA programming model, computations are done via several parallel kernels. Each kernel consists of a grid of thread-blocks (blocks for the sake of brevity), where each block is carrying a number of threads. Since no inter-block data dependencies exist in CUDA kernels, the blocks can be executed in any order.

CUDA Memory Model: Logically, in addition to the registers devoted to each thread, each of them possesses a private memory space. A shared memory space is shared between all of the threads in the same block. The global memory is accessible from all of the threads of all blocks.

CUDA Execution Model: When a CUDA kernel is launched on a GPU, the kernel blocks are first mapped onto the GPU SMs. Each SM is capable of performing a given number of blocks concurrently. If the number of mapped blocks exceeds the limit, the extra number of blocks stall until the in-flight blocks are completed. When a block starts executing on a SMs, it is divided into several warps that consist of 32 threads. Ready warps are scheduled onto the available intra-SM resources by warp schedulers. A warp may be stalled, for example, due to a memory reference or an instruction dependency. The number of in-flight warps in a SM is also limited and can further restrict the number of in-flight blocks. The number of warp schedulers and their scheduling policies are different from one GPU generation to another.

3.1 Cache Memory Hierarchy of Maxwell GPUs

In Maxwell GPUs (GM), L1 and Texture caches are integrated. Each SM has a total of 48 kB of L1/Texture cache divided into two 24 kB slices, where each slice is shared by a group of 64 processing cores. In addition to data caching, L1 cache is

also used for register spilling during the execution. L2 cache, consisting of a number of partitions/banks, is shared among all SMs, and all the global memory accesses go to the main memory through the L2 cache. Memory addresses are interleaved among the banks. Maxwell GPUs have several L2 partitions, where each partition consists of two 128 KB banks. An overview of memory hierarchy of Maxwell GPUs (GM) is presented in Figure 1 a). Moreover, the structure and mapping of L2 cache memory of GTX 970, which is used in our evaluations, are shown in Figure 1 b).

L1 caches bypass all the global write memory accesses (a write hit imposes an eviction), while the global read memory accesses can be optionally cached into or bypassed from the L1 depending on the bypassing strategy (not all the NVIDIA GPUs are capable of optional L1 global caching [35]). The bypassing strategy can be defined by compiler flags at the time of compilation.

Miss Status Holding Registers (MSHRs) are a set of registers that track the outstanding missed accesses. A missed access is first compared with the existing content of allocated MSHRs. Consequently, if the requested address is already present in a MSHR (requested by a prior access), the new access will be merged with the existing one, otherwise, a free MSHR will be assigned to the access. When the requested cache block arrives from the backing store, the reserved MSHR becomes free and cache will be filled using the arrived block. The number of MSHRs assigned to each cache is limited, and a reservation fail happens when a missed access does not obtain a MSHR. In this case, the missed access keeps trying to obtain a free MSHR in the next cycles, and the issuing load/store unit stalls during the reservation fail. In addition to the number of MSHRs, the maximum number of per-entry merges is also limited. MSHRs have an important role in delivering the non-blocking cache property, thereby highly affecting the performance of the memory [36]. Accordingly, MSHRs should be modeled as part of modeling the performance of cache memories. Further, it should be noted that the atomic instructions that are handled at L2 level are not considered in the present paper.

3.2 Reuse Distance Analysis

The aim of reuse distance (RD) analysis [4] is to profile the locality of applications. In addition, RD analysis can be used for modeling the cache performance of fully-associative caches with LRU replacement policy. The memory sequences (trace) of accessed cache blocks (or memory addresses) are analyzed to calculate their RDs. The value of RD for a given access to an address is calculated as the number of unique accessed addresses between the current and previous access. Although RD can be calculated with either of memory addresses or cache block granularity, the latter is considered in the present study.

Basically, the main property of RD analysis is its hardware independence. However, for a LRU cache with a given number of blocks, the resulting hit ratio of an application can be calculated based on the RD values of memory accesses. For a fully associative cache with K cache blocks, an access is a hit if its RD value is less than K, otherwise it is a miss. Based on the RD analysis algorithm, the RD value is

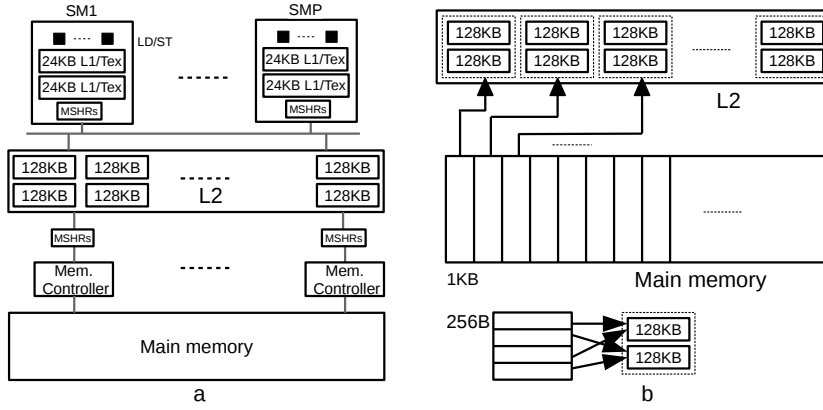


Figure 1. Overview of memory hierarchy of NVIDIA Maxwell GPUs [35], and two-level address mapping scheme of L2 in GTX 970

equal to infinity for the first access to an address, and therefore, access with infinite RD values represent the cold misses. When the intention of RD analysis is hit ratio calculation, a MRU stack is considered that its first block is the most recently used one. An array of counters (denoted by $C[K+1]$), which consists of $K+1$ counters, is used to count the hits and misses of the memory accesses, where $C[n]$ holds the number of accesses with RD values of n. $C[K]$ holds the number of accesses with RD $> K$ missing the cache. Given the counter values of a cache with K blocks, the hit ratios of caches with fewer K blocks (e.g., K') can also be calculated through summing up the first K' counters.

As for the set-associative cache memories, the same set of counters can be used for all the cache sets. In the present paper, RD analysis was used to model the performance of set-associative cache memories, and the same set of counters were employed for all cache sets. In Table 1, a typical example is given for RD calculation. In the case of caches with four blocks, the hit ratio equals 50 % without any capacity miss, while for a cache with two blocks, the hit ratio equals 25 %.

Step	0	1	2	3	4	5	6	7
Sequence	A	B	C	D	A	A	D	C
RD	∞	∞	∞	∞	3	0	1	2

The alphabet letters stand for the accessed cache blocks

Table 1. An example for RD analysis

4 RDGC PERFORMANCE MODEL

RDGC, short for reuse distance-based GPU cache, aims to model cache performance. In this method, cache memory hierarchies are analyzed through processing memory access sequences. To do so, the extracted memory trace of parallel blocks are converted into coalesced warp serial access and then analyzed by the RD algorithm that is presented in Section 4.4. In Figure 2, two components of RDGC, namely logical and physical models, are shown.

To analyze the cache performance, the logical model provides the memory access information which is independent of GPU. Then L1 and L2 cache memories are analyzed by the physical model based on the logical trace information along with the physical cache parameters and GPU specifications.

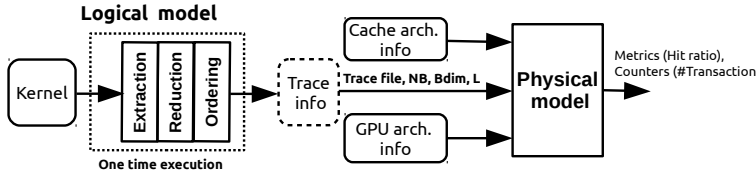


Figure 2. The RDGC components

4.1 Logical Model

To analyze the performance of kernels, the per-thread raw information is first extracted by the logical model. Then the trace file is reduced, and finally the accesses are ordered based on the logical execution model of CUDA. The three phases of the logical model are as follows:

Trace extraction. In the present study, the per-thread memory trace information is extracted through manual probing then executing the kernel. A considerable number of concurrent threads are performed by GPUs. Thus, recording the detailed information for each thread seems impractical and only represents the execution ordering on a specific device. Further, even recording the raw information of all GPU threads at once requires large buffers to store the recorded information temporarily during the trace generation. Consequently, to keep the trace file independent of GPU and to avoid large buffers, the following approach was adopted:

1. Only the raw memory access information (without time stamp) was recorded, and no information was recorded about the thread and instruction ordering and the block to SM mapping. Later, different block mapping and warp scheduling policies can be enforced by the physical model.

2. According to CUDA, blocks can be executed in any order, thus they were separately traced to further alleviate the buffer size. Recording the access information of all the blocks at the same execution run requires a considerable memory space. By separate block trace extraction, the kernel can be launched multiple times, where the information of only several blocks is recorded in each launch.

In Figure 3, the organization of the trace file is depicted, in which each line in the trace file contains the access information of one thread, denoted by ACC. ACC is a five-tuple set in the form of $ACC = \{AN, BID, TID, S, ADD\}$, where

- AN denotes a per-block access number assigned to each access of the block,
- BID is a unique block ID which is assigned to each thread block,
- TID represents the per-block thread identifier,
- S is a Boolean access specifier to define whether the access is a read or a write, and
- ADD denotes the accessed global memory address.

In Figure 3 an example is shown for M blocks and N threads per block ($BID = \{0, \ldots, M-1\}$, $TID = \{0, \ldots N-1\}$ where L denotes the maximum number of accesses within each thread ($AN = \{0, \ldots, L-1\}$). It should be noted that not all the threads within a block necessarily appear in the trace file, e.g., due to a warp divergence.

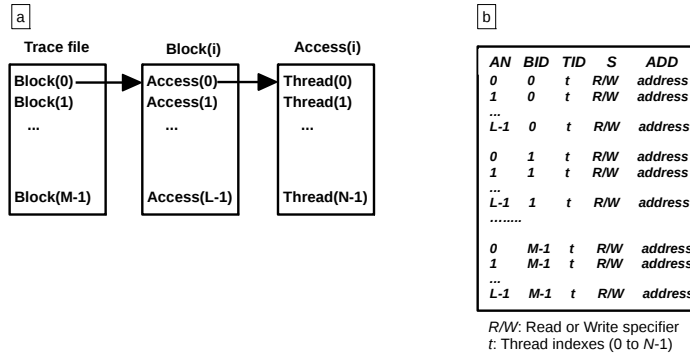


Figure 3. a) The hierarchical structure, b) and overview of the memory trace file

Trace File Reduction. For a kernel with high levels of memory access, the size of the trace file tends to grow rapidly. To alleviate the space overhead of a trace file and to accelerate its processing speed, the generated trace files are reduced through converting the thread access to the coalesced warp access. Moreover, the information of a warp access is stored by each line of the reduced trace file as $\{AN, BID, WID, S, NT, \{ADD\}\}$, where WID is the warp index that is calculated by dividing the thread indexes to the warp size (i.e., 32). Further, NT

denotes the number of active threads of the warp, and the accessed addresses are stored in $\{ADD\}$. As a result, the size of the trace files dropped by 2.3 times in the workload used in this paper.

Trace File Ordering. The memory accesses are ordered by the logical model without enforcing any physical limitations. No GPU related parameters, e.g., warp scheduling policy, are modeled at this step. In the logical model, it is assumed that the accesses to all blocks with the same access numbers (AN) can be executed in parallel with each other. The trace file is ordered according to ANs. Additionally, since the trace files are huge and stored on disks, their ordering is a time consuming operation. In addition to the logically-ordered trace files, grid dimensions (denoted by NB), block dimensions (denoted by $Bdim$), and maximum numbers of per-thread accesses (denoted by L) are generated by the logical model.

4.2 Physical Model

In addition to the trace file information, GPU and cache memory architectural information (listed in Table 2) are received by the physical model. The physical model calculates the MRU counters and then the L1 and L2 cache hit ratios and transaction counts are calculated from the MRU counters. Figure 4 shows the workflow of the physical model. In this figure, $C1R/C1W$ and $C2R/C2W$ denote L1 and L2 cache memory MRU counter arrays for read and write accesses, respectively. Further, the trace files are depicted as dashed rectangles, and the RD analysis, described in Section 4.4, is employed within the physical model to calculate the MRU counters.

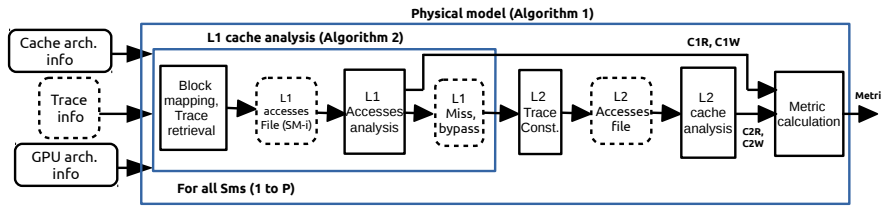


Figure 4. The structure of the physical model

The physical model operates according to Algorithms 1 to 3. As shown in Algorithm 1, the per-SM L1 cache memories are first analyzed (line 3), and the MRU counters (denoted by $C1R$ and $C1W$, for read and write accesses, respectively) are updated accordingly. $C1R$ and $C1W$ are counter arrays with K_1 elements (K_1 is the L1 cache associativity), used cumulatively for all L1 caches. After analyzing the L1 caches, the L2 cache trace file is constructed through retrieving and analyzing the missed or bypassed L1 accesses (line 5) to calculate the L2 MRU counter arrays (denoted by $C2R$ and $C2W$) (line 6). Finally, the output metrics (see Section 5) are calculated based on the MRU counters.

Parameter	Value/Options	Comment
Capacity	S	Capacity of the cache
Associativity	K	Cache's associativity
Block size	B	Block size in bytes
Parallelism	Partition	Number of cache Partitions
	Bank	Number of cache banks per each partition
	Port	Number of ports per each cache bank
Indexing function	MOD	Modulo indexing
	SMODn	Shifted MOD: $[i+m+n,\ldots,i+n]$ index bits used instead of $[i+m,\ldots,i]$ (m, n are #shifts and index bits [37])
	PRI	Prime modulo Indexing [33]
	XOR	Xor based indexing
Replacement[2]	LRU	Least Recently Used
	LFU	Least Frequently Used
	FIFO	First In First Out
	RANDOM	Random
Bypassing policy	WON	Writes ON, bypass all the write accesses
	RWON	Reads and Writes ON, all accesses are bypassed
	OFF	Bypassing disabled
MSHR	#MSHR sets	Number of MSHR sets
	MSHR Size	Number of MSHRs per each MSHR set
	Max#Merges	Maximum number of merges per MSHR
Resources	P	Number of SMs of the GPU
	n_scheduler	Number of Schedulers per SM
	MAXCW	Max number of in-flight warps
	MAXCB	Max number of in-flight blocks
Blocks-SM mapping	RR	Round-Robin
	BPART1/2	Partitioning, partitions of four/eight blocks
	RAND	Random

[1] Cache parameters are defined for both the L1 and L2
[2] For non-LRU policies, only hit ratio is calculated and the counters do not contain the corresponding RD values.

Table 2. Cache and GPU related inputs to the physical model[1]

4.3 L1 Cache Analysis

The L1 cache analysis algorithm is shown in Algorithm 2. In this algorithm, first, the assigned blocks to the SM are defined based on the given mapping policy (line 1). Then, the access information of the blocks is retrieved from the trace file and stored in a file, called $L1\_access$ file (line 3), which is analyzed according to the execution model of the GPU to calculate the MRU counters. In the algorithm, B is an array that contains the block indexes of the SM, and BSM denotes the number of blocks

mapped to the SM. As noted before, due to the resource limitations, the number of warps and blocks that can be simultaneously executing on each SM is limited. The maximum in-flight warps and the maximum in-flight blocks per each SM, which are GPU specific parameters, are denoted by $MAXCW$ and $MAXCB$, respectively. Further, CB denotes the number of in-flight blocks on a SM that is defined based on:

1. two kernel-related parameters: grid dimension (denoted by NB) and block dimension (denoted by $Bdim$);
2. two GPU-related parameters: maximum number of in-flight blocks (denoted by $MAXCB$) and maximum number of in-flight warps (denoted by $MAXCW$).

Algorithm 3 is used to define both CB and the number of iterations (denoted by $Nitr$) required to analyze all the blocks. The number of blocks of the SM (denoted by BSM), is defined based on the chosen blocks to SM mapping policy. In Algorithm 2, when CB and $Nitr$ are defined through invoking Algorithm 3 (line 4), the analysis is performed $Nitr$ times, each time for the maximum of CB blocks, through retrieving and processing the information of the in-flight blocks. In each iteration, the order of accesses is the very same order defined by the logical model.

To analyze each access within an inflight-block, the information with memory address granularity is converted to a coalesced cache block access. The coalesced access information of one or more warps (depending on the number of schedulers per SM, $n\_scheduler$) is stored within a list (denoted by W), and then the RD analysis is applied to W (line 11). Once all in-flight blocks are analyzed, they will be retired and a new set of blocks (if any) will be processed (line 5 to 15) to analyze all BSM blocks of the SM.

Due to space limitation and its similarities to L1 analysis, the L2 analysis algorithms are not covered here.

Algorithm 1: Physical model

```
Input: Trace, cache parameters, GPU specification
Output: Metrics
  initialize();
  for sm := 1 to P do
    L1_cache_analysis(sm, Trace); /* Update C1R, C1W. Algorithm 2 */
  end for
  L2_trace_construction();
  L2_cache_analysis(); /* Calculate C2R, C2W */
  Metrics = calculate_metrics(C1R, C1W, C2R, C2W);
```

4.4 RD Calculation for Parallel Cache Memories

In this section, an RD analysis algorithm is proposed for parallel caches with multiple banks and access ports. Since GPU hardware parameters affect the resulting RD

Algorithm 2: L1 cache analysis

Input: sm, Trace,
Output: Update $C1R$, $C1W$
1: $\{B, BSM\} = block\_mapping(sm, NB)$; /* Define the blocks of the SM */
2: $WpB \leftarrow \lceil \frac{Bdim}{Wsize} \rceil$
3: $L1\_accesses = trace\_retrieval(B, BSM, \text{Trace})$;
4: $\{CB, Nitr\} = define\_concurrent\_blocks(WpB, BSM)$; /* Algorithm 3 */
5: **for** $i := 0$ to $Nitr - 1$ **do**
6: $get\_inflight\_trace(B, L1\_accesses)$; /* retrieve access info of the in-flight blocks */
7: **for** $j := 0$ to $L - 1$ **do**
8: **for** $k := 0$ to $CB - 1$ **do**
9: **for** $l := 0$ to $\lceil WpB/n\_scheduler \rceil - 1$ **do**
10: $W = create\_access\_list(i, SB[k], l)$;
11: $RD\_profile(W)$; /* Section 4.4 */
12: **end for**
13: **end for**
14: **end for**
15: **end for**

Algorithm 3: Defining concurrent blocks

Input: WpB, BSM
Output: $CB, Nitr$
1: $CB \leftarrow BSM$
2: **if** $BSM > MAXCB$ **then**
3: $CB \leftarrow MAXCB$
4: **end if**
5: **if** $WpB \times CB > MAXCW$ **then**
6: $CB \leftarrow \lfloor \frac{MAXCW}{WpB} \rfloor$
7: **end if**
8: $Nitr \leftarrow \lceil \frac{BSM}{CB} \rceil$
9: **return** $CB, Nitr$

values, it is no longer a hardware independent algorithm. As explained above, the physical model properly generates the warps to cache access sequence. Therefore, the cache reference sequence is known in this stage, however, the sequence is not pure serial and satisfies cache level parallelism (several warp schedulers issue coalesced accesses). The coalesced accesses are mapped onto the cache banks and ports. The proposed RD analysis method is similar to the method introduced in [8]. The following summarizes the differences of the present study with the mentioned work.

- Nugteren et al. modeled serial caches. However, cache level parallelism can change the achieved RD profile (see Section 5.3.3) and the present work included cache level parallelism.

- The authors only modeled L1 cache while both L1 and L2 caches are included in our model. Further, more cache related organizational parameters are investigated in our work.
- Since L1 cache bypasses the write accesses, Nugteren et al. only considered read accesses. However, for write-evict policy, ignoring the write accesses can cause considerable errors in write intensive applications. In the present study, both read and write accesses are included and L1 either caches read accesses (enabled) or bypasses read and write accesses (disabled). When enabled, L1 follows the write-evict policy [34].
- Nugteren et al. assumed that a reservation fail cancels the failed access while other accesses of the same warp, possibly from later instructions, can proceed. This means that load/store instructions may be executed out of order, which is not realistic. In this paper, like some other researchers [36], a reservation fail stalls the warp until all the accesses of the warp are serviced.
- In the mentioned research, the notion of latency miss is introduced to count the event in which a miss encounters a pending previous miss to the same cache block (which exists in a MSHR). In this article, since such requests are merged into the existing MSHR, this parameter is equal to the number of merged requests.
- The probabilistic latency model introduced by the authors can repeatedly change the access order while, as described in the following sub-section, the adaptive latency model can produce smoother and more realistic latency values for the missed accesses.

In Table 3, an example of RD calculation is shown for three warps that access {A, B, C, D}, {E, F, G, H}, and {A, D} cache blocks, respectively. Each cache block is mapped to one of the cache banks. The assumed cache has two banks (B_0, B_1), each having two ports (P_0, P_1), and two MSHRs are shared between the banks. Further, it is supposed that A, C, E, and G are mapped to $B0$ and the other blocks to $B1$. Note that RD is calculated for each bank separately. The first row of the table shows the steps of RD calculation. The next three rows demonstrate the accessed blocks that mapped to each cache bank/port and their corresponding RDs. In addition, the fifth row shows the number of free MSHR entries and the next two rows illustrate the corresponding status of each access (hit (h), miss (m), or reservation fail (rf)). Finally, the last row represents the updated cache blocks which is done by the arrived blocks from the backing store. It is assumed that there are two warp schedulers that coalesce and issue the warp accesses.

Each step of RD calculation includes two phases. In the first phase, the requested blocks are mapped onto the cache banks (according to a given mapping scheme) and access the banks through the available ports (if any). If an access misses the cache, a MSHR entry is reserved when possible, otherwise (denoted by rf in Table 3), the failed access will keep trying to reserve an MSHR in the subsequent steps.

In the second phase, the state of the cache is updated by the cache blocks arriving from the backing store, their assigned MSHR entries become free and update the

cache state so that they are available in the next steps. It should be noted that a hit access also causes some cache updates. In Table 3, the latency of missed access equals two steps. Hence, the requested block by a missed access in the i^{th} step arrives at the end of the step i+1 and updates the cache state. As a result, this block will be available from the step i+2 forward. It should be noted that 'step' denotes a virtual notion and is not the same as clock cycle. It can be used, nevertheless, as a performance criterion in RD calculation.

The number of cache banks and access ports of each bank can alter the resulted RDs, thus their inclusion in the model is necessary. In the proposed algorithm, to define the exact warp sequence, the following assumptions are considered:

- Warps with smaller indexes have higher priorities in accessing banks and MSHRs.
- Warp schedulers stall until all issued accesses are resolved [36].
- In each step, multiple accesses can be inserted to or removed from the MSHRs.
- Multiple cache blocks can arrive from the backing store within the same step and fill the cache at the end of that step. In this case, the cache state is updated within the same order that the arrived access has been inserted into the MSHRs. This order affects the subsequent RD calculations.

Step		0	1	2	3	4
B_0	P_0	A	*E*	E	–	A
	P_1	C	*G*	G	–	–
B_1	P_0	*B*	B	*F*	F	D
	P_1	*D*	D	*H*	H	–
RD		∞	–	∞	–	3
		∞	–	∞	–	–
			∞	–	∞	2
			∞	–	∞	–
#Free MSHR		0	0	0	0	2
Status, B0		m	rf	m	–	h
		m	rf	m	–	–
Status,B1		rf	m	rf	m	h
		rf	m	rf	m	–
Update		–	A	B	E	F
		–	C	D	G	H

Table 3. An example of RD calculation in Parallel cache memories

4.4.1 The Latency Model for RD Calculation

In the RD analysis algorithm, an appropriate model is required to properly define the latency of missed accesses, based on which the cache state is updated. The values of access latency within a real GPU depend on many parameters, e.g., the instruction

mix, memory access pattern, and the L1-L2 and L2-main memory bandwidths. The probabilistic model used by Nugteren et al. can produce substantially different latency values for two close accesses and even may re-order them, thus we ignore this model. Instead, two types of latency models are tested in the present study. The first latency model sets the latency of the missed accesses to a fixed value, whereas the second is an adaptive model that calculates the latency values based on some dynamic run-time statistics and can provide smoother and more realistic latency values than a probabilistic model. For L1, the latency is calculated by the adaptive model as $K_1 + K_2 \times \frac{\#MSHR\_Busy \times \#active\_SMs}{L2P}$, where K_1 and K_2 are constant values, $\#MSHR\_Busy$ represents the number of outstanding misses, $\#active\_SMs$ is the number of active SMs during the execution, and $L2P$ denotes the L2 cache parallelism. The constant values should be defined according to the GPUs data transfer bandwidths. It should be noted that the possible bottlenecks are ignored in the proposed model at L2-main memory transfers. A similar model can be derived for L2. We performed an analysis to investigate the effects of the mentioned latency models on the resultant performance parameters and presented the analysis results in Section 5.1.

4.4.2 L1 and L2 Cache Parallelism Modeling

In the present paper, the default configuration of L1 caches were a double-ported single-bank caches with a set of 32 MSHRs and the maximum number of eight merges per entry. Further, L2 cache was modeled based on the organization shown in Figure 1 (b), and four MSHR sets with 32 entries were used for L2. The first three MSHRs sets were assigned to the first six L2 partitions (one MSHR set shared between two partitions) and the last MSHR set was assigned to the last partition.

5 RDGC EVALUATION

In the present study, mainly, Polybench/GPU benchmark suite [5] is used as the main workload. In addition, several cache intensive kernels were included from Rodinia [6]. Polybench/GPU kernels immensely rely on the hardware managed cache memories thus put more pressure on the cache hierarchy, which is the focus of this paper. On the other hand, most other benchmarks heavily used shared memory and thus most of the data transfers are handled by the shared memory. Consequently, the hardware managed caches are only used to transfer the required data to shared memory. As a result, such benchmarks may fail to properly stress the L1 cache and especially L2 cache, which is an order of magnitude bigger than L1 caches. It should be noted that, none of the used benchmarks utilized atomic instructions, and texture caches. The benchmarks with their main specifications are listed in Table 4. RDGC evaluation was performed within three steps. In the first step, the latency models introduced in Section 4.4.1 were tested (Section 5.1). In the next step, RDGC was validated by comparing its outputs with the values recorded by the performance counters (accessed through NVPROF) on two GPUs including

a Kepler GT 740M and a Maxwell GTX 970 (Section 5.2). In the last step, different cache organizational parameters, including cache capacity, associativity, block size, mapping functions and replacement policies, were evaluated. In addition, multiple blocks to SM mapping policies were evaluated and, finally, the effects of multiple L2 cache parallelism levels on the achieved RD values were analyzed (Section 5.3).

The outputs are provided by RDGC as several metrics. It should be noted that in GT 740M GPU, L1 is disabled for both load and store accesses. Further, in GTX 970 GPU, the texture cache (denoted by tex in the figure) is the same as L1 cache. As mentioned earlier, two compiling options are available for GTX 970: "`-Xptxas -dlcm=cg`" option to disable L1 and "`-Xptxas -dlcm=ca`" option to enable the L1, which in this case L1 only caches the read accesses. Although NVPROF provides a metric to represent the texture cache hit ratios, this counter also counts the other non-workload accesses, e.g., register spilling. Hence, this value is not the exact value of the hit ratios of the requested workload data. In this work, L1 hit ratios are calculated indirectly from other counters. The same phenomenon also occurs at L2 level. Typically, since L2 is significantly greater than L1 and the fact that most of the non-workload traffics are filtered at L1, the resultant errors are likely to be negligible. The brief explanation of the output parameters calculated by RDGC is as follows:

- $L1\_R\_Hit$ is the read hit ratio of L1, when L1 is enabled.
- $L2\_Hit$ is the hit ratio of L2 when L1 is enabled.
- $L2\_Hit\_L1B$ is the hit ratio of L2 when L1 is disabled (bypassed).
- $L1\_R\_Trans$ is the read transaction count of L1, when L1 is enabled.
- $L2\_Trans$ is the transaction count of L2 when L1 is enabled.
- $L2\_Trans\_L1B$ is the transaction count of L2, when L1 is disabled.

5.1 Latency Models Evaluation

In this section, the latency models, including the fixed and adaptive models introduced in Section 4.4.1, are tested to reveal the effects of latency on L1 cache performance. The analyzed system has eight SMs, 32 KB of four-way set-associative LRU L1 caches with 32 B blocks, and a set of 32 MSHRs per L1 with maximum of eight per-entry merges. Figure 5 shows the effects of latency on a) hit ratios, b) reservation fails, c) MSHR address merges, and d) steps variations in RD calculation. For the adaptive model, four different values of (K_1, K_2) are tested including $(1, 0.125)$, $(2, 0.25)$, $(4, 0.5)$ and $(8, 1)$ which are denoted by A1 to A4, respectively. Furthermore, to observe the RD calculation performance, the variability of steps in RD calculation is also given in the figure with respect to the number of step counts of the fixed latency with value of one. As can be seen, by changing the latency, the resultant merged and reservation fails are significantly changed, however, hit ratio is not witnessed extreme changes. In the rest of this paper, A2 model is used (the constant values are defined based on the GPU specifications).

Application	Size (N)	Kernels	L	RDGC slowdown (10^3)	GPGPU-Sim slowdown (10^3)
2DCONV†	4 096	2DCONV	10	212	3 084
2MM†	384	K1	$3N$	59.2	1 068
3DCONV†	256	3DCONV	$12N$	69.2	3 409
ATAX†	4 096	K1,K2	$1+3N$	56.4	6 546
BICG†	4 096	K1, K2	$1+3N$	45.6	2 663
CORR†	256	CORR	$(3N+2)N+1$	2.7	190
COVAR†	256	COVAR	$(3N+2)N$	2.6	219
FDTD†	2 048	Step1, 2, 3	6, 4, 6	58.4	3 887
GESUMMV†	4 096	GESUMMV	$8N+2$	64.8	5 043
GRAMSC.†	128	K3	$7N^2$	1.5	91
MVT†	4 096	K1, K2	$3N+1$	14.4	4 810
SYR2K†	256	SYR2K	$5N+1$	80.3	9 900
SYRK†	256	SYRK	$3N+2$	57	4 085
BP‡	262 144	K1, K2	16	28.6	800
CFD‡	0.2 M	Flux	83	51.7	1 626
HSPOT‡	1 024	HSPOT	3	14.1	817
NW‡	4 096	K1, K2	8 384	13.6	819
SRAD_V2‡	2 048	K1, K2	12	11.4	639
Average				**46.9**	**2 761**

† From Polybench/GPU [5], ‡From Rodinia V3.1 [6]

Table 4. Polybench/GPU and Rodinia benchmarks, specifications, and slowdowns of RDGC and GPGPU-Sim with respect to the executions performed on a GTX 970 GPU

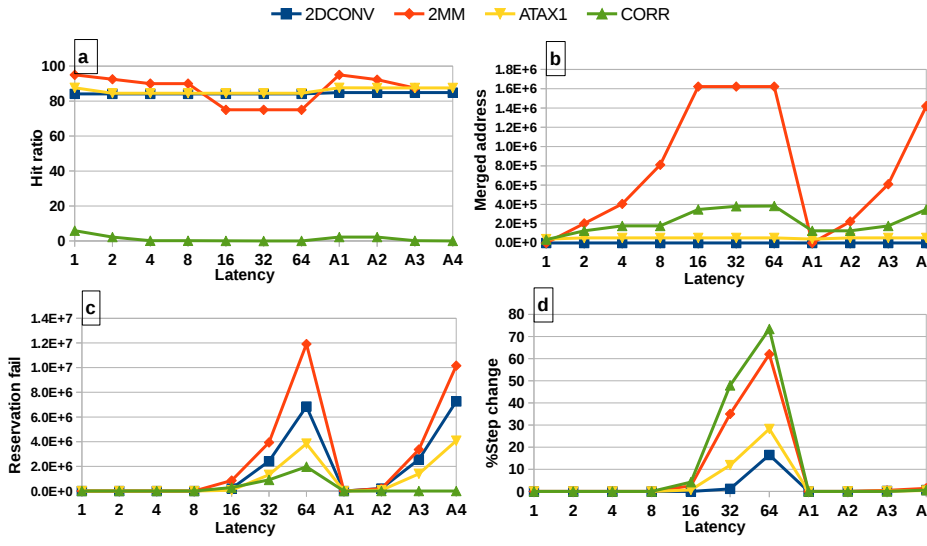


Figure 5. The effects of latency on different parameters in the model

5.2 RDGC Validation

RDGC is validated through comparing its outputs with the results provided by NVPROF for profiling the same workload on a Maxwell GTX 970 GPU and a Kepler GT 740M. Table 5 gives the parameter values used by RDGC. These values are selected based on the available NVIDIA documents [35] and the findings of previous studies [37]. However, some important cache parameters are neither reported by NVIDIA, nor discovered by the research community, e.g., L2 mapping function, the number of L1/L2 access ports, and L2 replacement policy. The analysis results of GTX 970 are shown in Figure 6 (hit ratios), Figure 7 (transaction counts). In addition, Figure 8 gives a comparison between the profiling results on a GT 740M and analysis results provided by RDGC.

Parameter	GTX 970	GT 740M
P	26[1]	2
L1 (¡S_1,K_1, B_1, Map., Repl.¿)	24 KB, 192, 32 B, XOR, LRU	–
L2 (¡S_2,K_2, B_2, Map., Repl.¿)	1792 KB , 8, 32 B, XOR, LRU	512 KB, 8, 32 B, XOR, LRU
Blocks to SM mapping	RR	RR

[1] GTX 970 has 13 SMs and each SM has two 24 KB L1 cache partitions, hence P and S_1 were set to 26 and 24 KB, respectively.

Table 5. The main configuration parameters of RDGC

5.2.1 The Physical Model Slowdowns

The physical model slowdowns were calculated through dividing their execution times measured on a system with Ubuntu 12.04 OS, Core i5 CPU, 4 GB of RAM, by the kernel execution times measured on a GTX 970 GPU (CUDA 7.0). All the kernel data transfer times are excluded. Further, the time overheads of the logical model were not included in the slowdown calculations. As shown in Table 4, the physical model had an average slowdown of 47K times, where 3DCONV had the highest slowdown (212 K times) as opposed to Gramschmidt with the lowest slowdown (1 504 times). It is worth mentioning that the performance of RDGC can be enhanced by employing some techniques such as parallel execution and statistical sampling methods [27]. The average slowdown of GPGPU-Sim measured 2 761 K (power simulation and visualizer was disabled). Therefore, RDGC (taking several minutes per application) is 59 times faster than the cycle-accurate simulation, while most of the applications take several hours to be simulated by GPGPU-Sim.

5.2.2 Discussion

According to the findings presented in Figures 6 to 8, the model has a fair accuracy in predicting the hit ratios and transaction counts. For the Maxwell GPU, The average absolute errors of $L1\_R\_Hit$, $L2\_Hit$ and $L2\_Hit\_L1B$, were 3.72 %, 4.5 %, and 4.52 %, respectively. Further, the average error of $L1\_R\_Trans$, $L2\_Trans$ and $L2\_Trans\_L1B$ were 15.0 %, 11.9 %, and 7.6 %, respectively. In the case of the

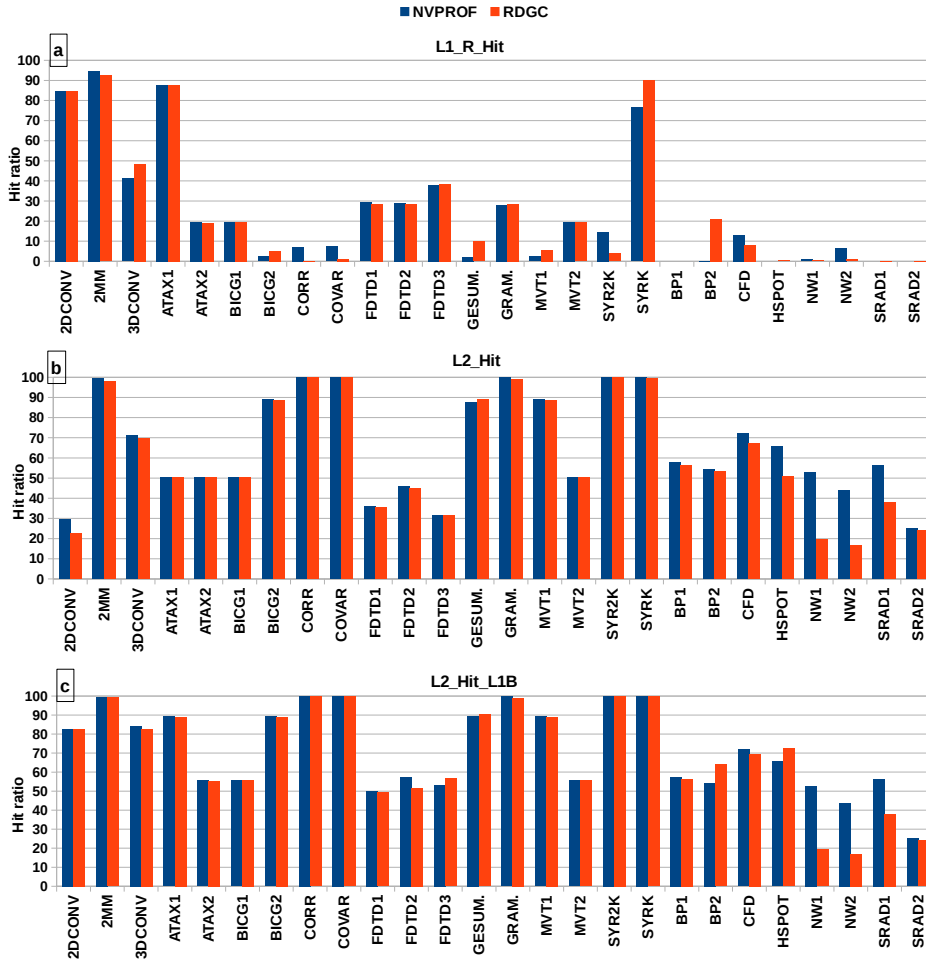


Figure 6. RDGC correlation with Maxwell GPU (GTX 970) (hit ratios)

Kepler GPU, the results has the average error of 5.4 % for $L2\_Hit\_L1B$ and the average error of 5.6 % were observed for $L2\_Trans\_L1B$. Furthermore, GPGPU-Sim has an average error of 23.3 % and 11.7 % for $L1\_R\_Hit$ and $L2\_Hit$, respectively. Note that any error in $L1\_R\_Hit$ may cause a high amount of error in $L2\_Trans$ (and $L2\_Hit$). Moreover, since the size of the L1 cache is limited, $L1\_R\_Hit$ is sensitive to L1 cache parameters. Moreover, in some kernels which extensively use shared memory, if the content of shared memory is spilled to the global memory, some transactions are generated at L1 and L2 caches to carry the spilled data. In our model, this phenomenon is ignored at L2 which can cause some error. Nevertheless, this phenomenon has only occurred in NW benchmark.

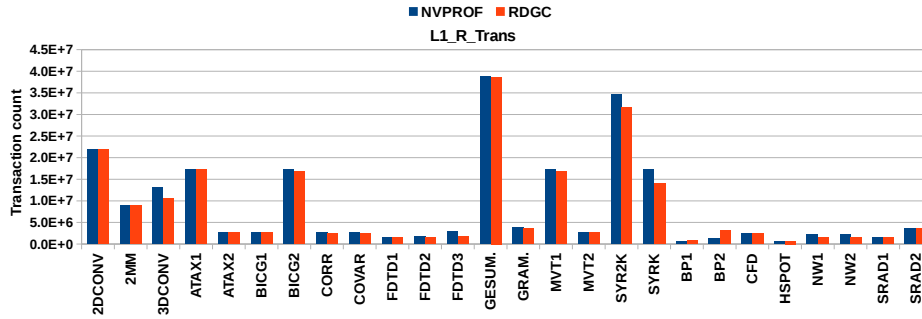


Figure 7. RDGC correlation with Maxwell GPU (GTX 970) (transaction counts)

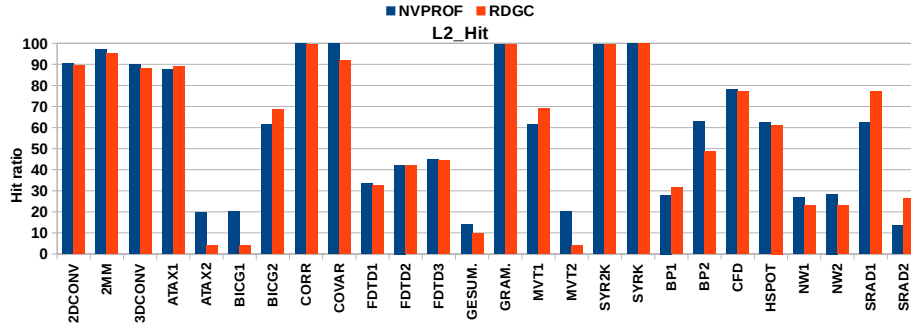


Figure 8. RDGC correlation with Kepler GPU (GT 740M)

Table 6 compares the RDGC model and the work of Nugteren et al. [8].

5.3 The Architectural Space Exploration of GPU Cache

In this section, different cache design parameters are explored. Only 2DCONV, 2MM, ATAX1 and CORR kernels were included in the evaluation. The selected kernels have diverse specifications in terms of their maximum number of per-thread accesses, grid dimensions, and block dimensions. The baseline GPU parameters applied for the simulations include 8 SMs, 32 KB L1 4-way set-associative in the form of a double-ported cache bank, 1 024 KB L2 8-way set-associative with four partitions and two 128 KB banks per partition, PRI mapping for L1 and L2, LRU replacement for L1 and L2, 128 B cache block size, and RR thread block to SM mapping policy. In total, 263 simulations were performed.

5.3.1 Analyzing the Effects of Cache Organizational Parameters

Cache size: The results of different cache sizes are shown in Figure 9. As can be seen, even small L1 caches result in large hit ratios in 2MM. In addition,

Specification	RDGC	Nugteren et al.
Coverage	L1 and L2	L1
Modeled GPU	Kepler (GT 740M), Maxwell (GTX 970)	Fermi (GTX 470)
Cache parallelism	multiple partitions, banks, ports	None
Mem. latency model	Adaptive	Probabilistic
Cache bypassing	Coarse	–
parameters	Hit ratio, Transaction count	Miss rates
Cache parameters	Capacity, associativity, block size	Capacity, associativity block size
Cache replacement	LRU, LFU, FIFO, Random	LRU
Mapping function	PRI, XOR, MOD, SMOD	Custom XOR
Block to SM mapping	RR, Partitioned, Random	RR

Table 6. Comparison of RDGC with the work of Nugterent et al. [8]

2DCONV highly benefits from increasing the L1 cache capacity. For instance, doubling the size of the 8 KB L1 cache led to increasing the hit ratio by 78 %. (Figure 9 a)).

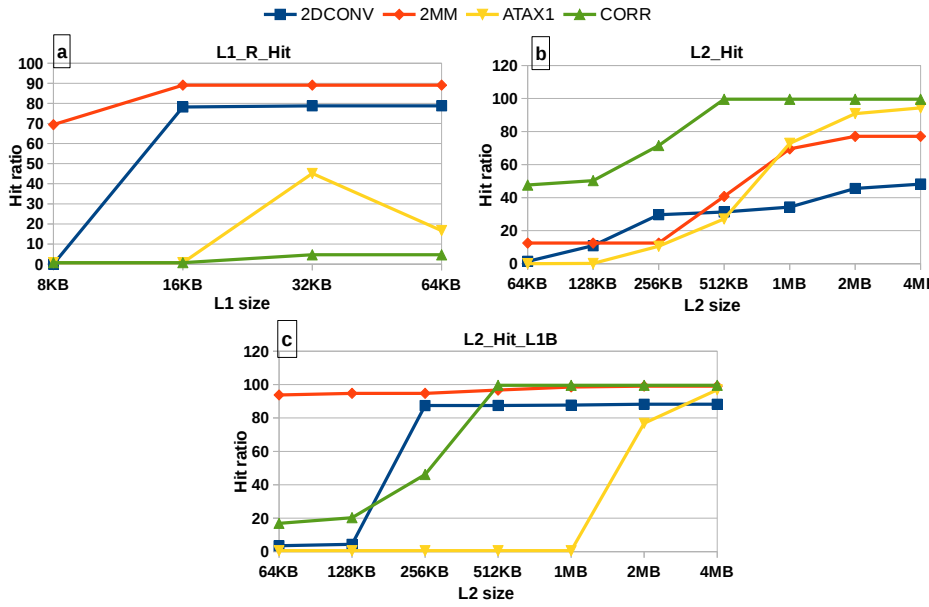


Figure 9. L1 and L2 performance for different cache sizes

Cache Mapping Function: In Figure 10, the *L1_R_Hit*, *L2_R_Hit*1 and *L2_R_Hit*2 metrics are presented for different cache mapping functions. As it can be observed, PRI functioned better than others, while the resultant hit

ratios significantly declined in several cases in MOD and shifted MOD mappings.

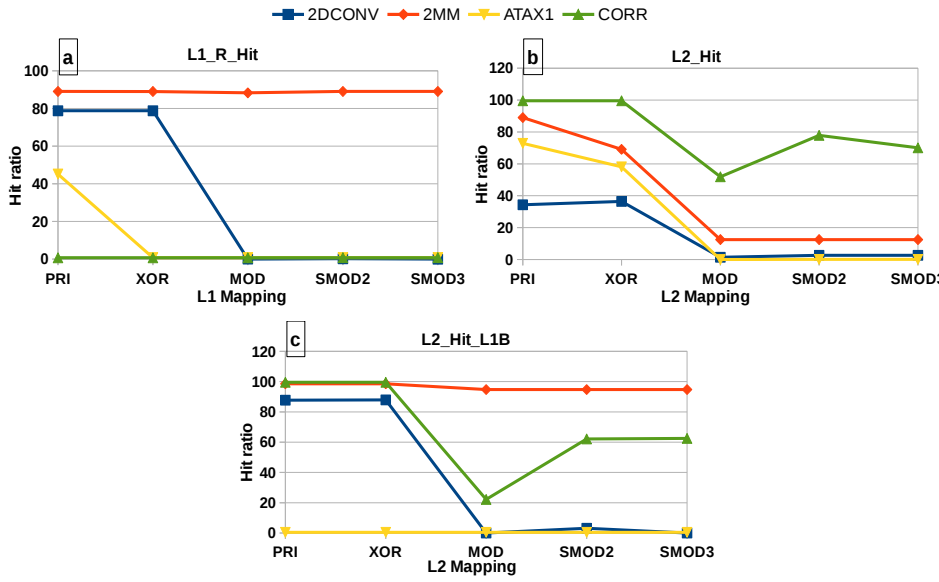


Figure 10. L1 and L2 performance for different cache mapping functions

Replacement Policy: In Figure 11, the metrics for different cache replacement policies are shown. For L1 cache, the performance of CORR was enhanced by LFU, whereas the performance of other three kernels was reduced. Further, ATAX1 achieved the best performance with random replacement in both L1 and L2 caches. Additionally, the performance of 2DCONV diminished as a result of employing LFU, but remained the same for other policies. On the other hand, 2MM showed less sensitivity to replacement policies than the other kernels. Overall, cache replacement policy is an important organizational parameter in cache memories, especially in L1 cache. Since no replacement policy functions the best all the time, employing the adaptive replacement policies is a promising approach.

Cache Block Size: In Figure 12, the resultant performance of different cache block sizes in L1 and L2 caches are illustrated. Except for ATAX1, the performance of L1 did not significantly change. Note that when L1 hit ratio is high, any small changes in L1 hit may result in radical changes in the transaction counts and hit ratios of L2 caches.

Cache Associativity: In Figure 13, the resultant metrics for different associativity (32 KB L1 and 1 MB L2) are shown. Figure 14 shows the RD profile for L2 cache including both read and write transactions. Note that the RD8+ in this

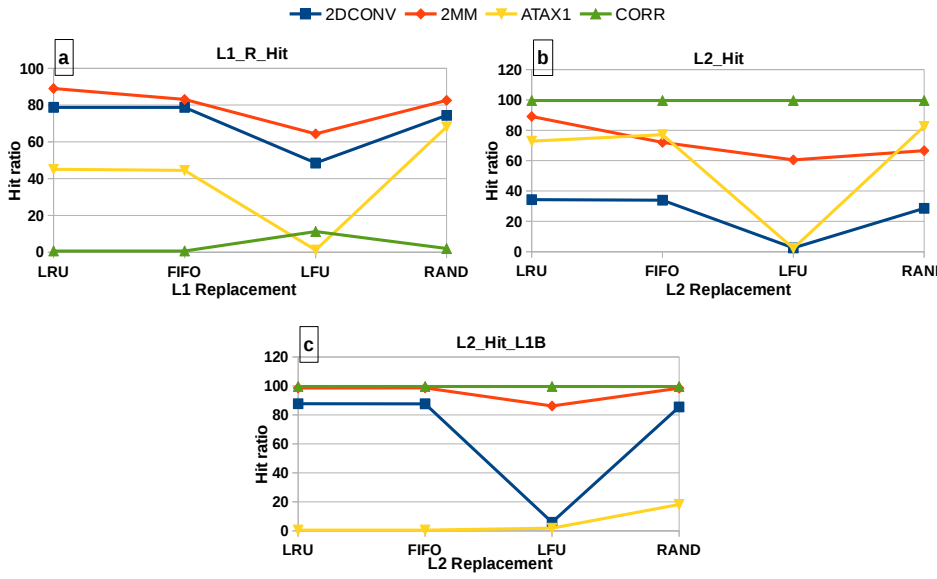


Figure 11. L1 and L2 performance for different cache replacement policies

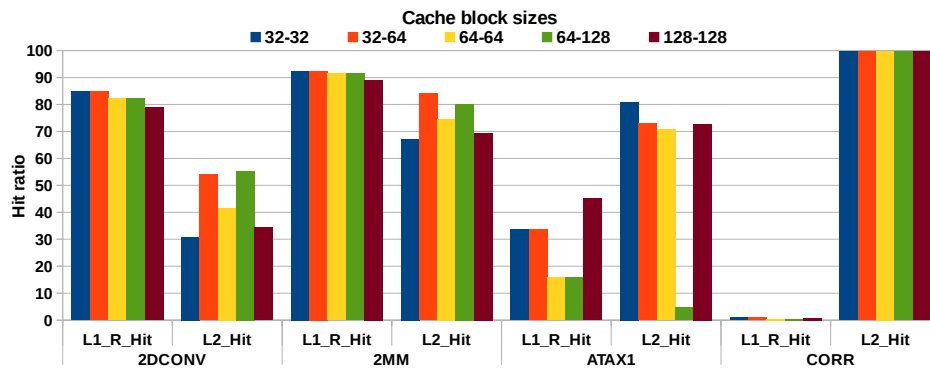


Figure 12. Cache performance for different cache block sizes

figure shows the missed Access. Moreover, RD profile is very helpful for performance analysis and characterization of application data reuse in many memory intensive GPU applications [28].

5.3.2 Blocks to SM Mapping

In Figure 15 the results of different CUDA thread blocks to SM mappings (see Table 2) are shown. Since CORR has only eight blocks, its results are not presented here. The results indicated that L1 cache performance was more sensitive to the

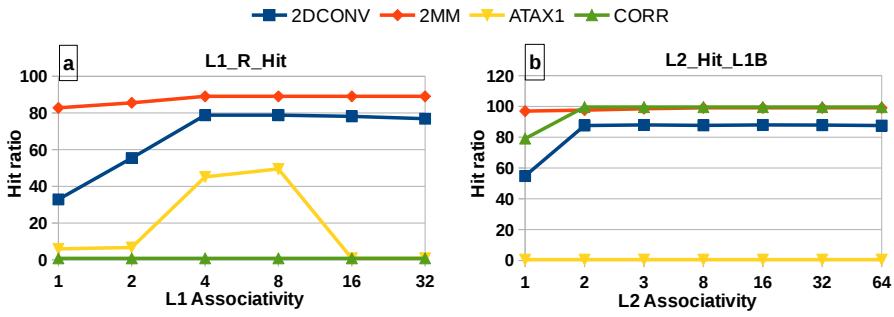


Figure 13. L1 and L2 performance for different cache associativity

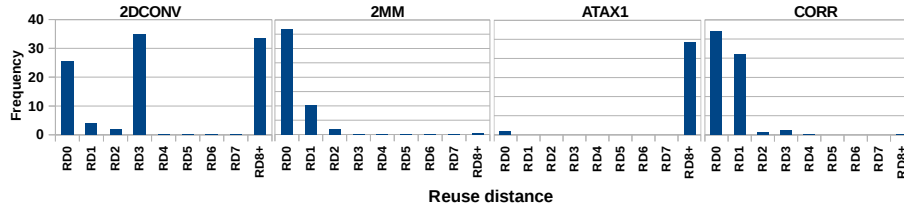


Figure 14. RD profile for L2 Cache (RWON)

blocks to SM mapping policies than L2. For example, ATAX1 achieved 65 % and 62.5 % of hit ratios for PART1 and PART2 and 45 % and 42 % of hit ratios for RR and Random block mapping policies, respectively.

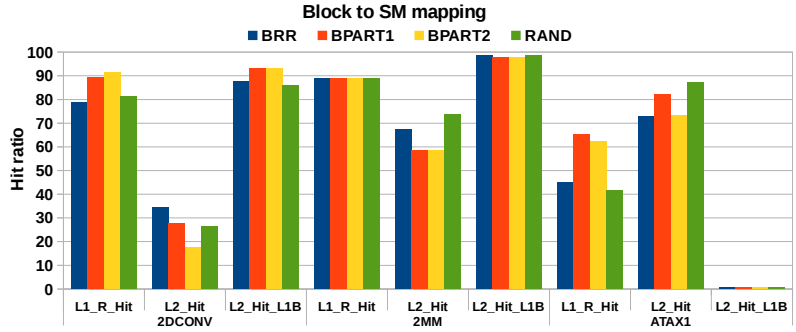


Figure 15. Cache performance for different thread blocks to SMs mapping policies

5.3.3 L2 Cache Parallelism Modeling

In this section, the performance of L2 cache (L1 disabled) for different cache parallelism levels are presented (see Figure 1 b)). Since GESUMMV has a considerable

number of transactions, it is included in this experiment. The two-level interleaving scheme was used for address mapping. In Figure 16, calculated MRU counters are presented. As it can be observed, cache parallelism changes the achieved reuse distance values, thus it is necessary to include the effects of cache parallelism in RD calculation. Even in 2DCONV and 2MM kernels that the hit ratios remained the same with different L2 parallelism levels, the reuse distance values were changed. These changes show that for bigger workload sizes or cache capacities, cache parallelism can alter the resultant hit ratios. Finally, Figure 17 shows the change in step counts as a function of L2 cache parallelism level.

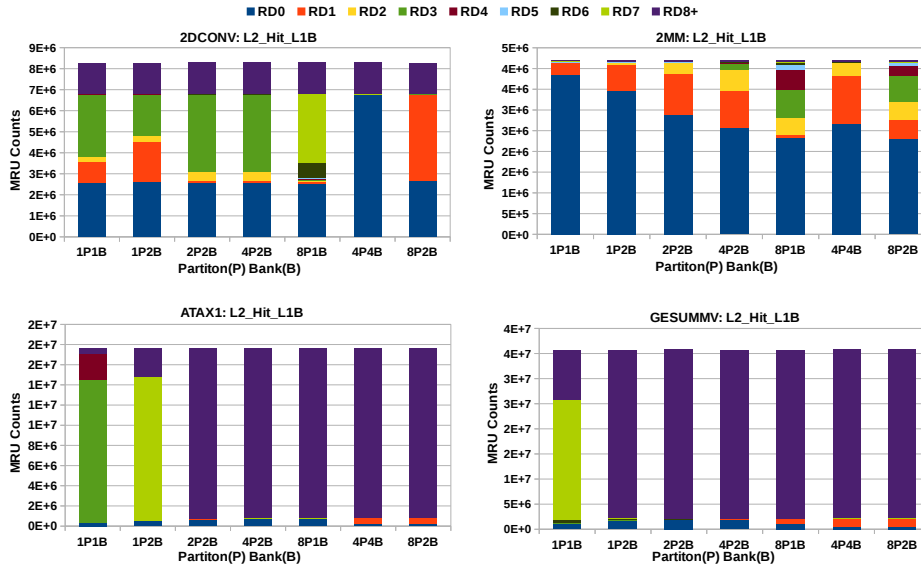


Figure 16. Reuse stack distances for different L2 parallelism levels (partitions (P), banks (B))

5.3.4 Discussion

While exploring the architectural space of cache memories through cycle-accurate simulation is extensively time-consuming, RDGC offers a more time-efficient approach to profile data locality and to model cache performance. However, RDGC is not a replacement for cycle-accurate simulation. Instead, it can be employed to narrow down the vast architectural space of GPU cache memories by analyzing different candidates for cache memory organization. RDGC can also be used by application developers to profile the data reuse. In addition, RDGC is agile to changes and can be modified to analyze caches in newer GPUs, without spending much effort.

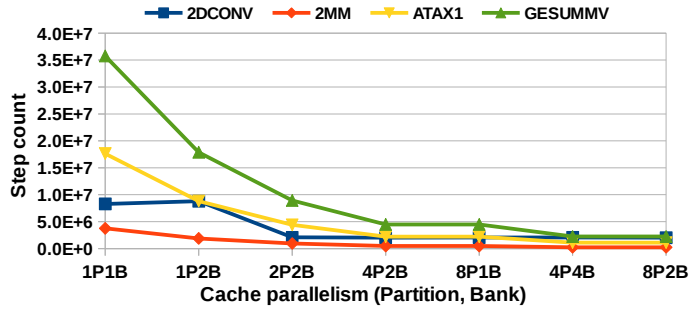


Figure 17. Step count trend for different L2 parallelism levels (partitions (P) and banks (B))

6 CONCLUSION

GPU architects and application developers need to analyze cache memory performance for different cache design parameters and different application configurations. Analyzing the performance of cache memories is a time-consuming task, especially for GPUs that execute threads in massive parallelism. This paper proposes a performance analysis approach, called RDGC, that applies reuse distance analysis to analyze the performance of GPU cache memory hierarchy. The evaluation results show that RDGC has fair performance and accuracy: 59 times speedup over GPGPU-Sim and an absolute error of 3.72 % and 4.5 % for L1 and L2 cache read hit ratios. Further, RDGC facilitates the architectural space exploration of GPU cache hierarchy. Different cache architectural parameters were modeled including: capacity, associativity, mapping (indexing), block size, replacement policy, and bypassing. In addition, the effects of cache parallelism (multi-bank and multi-port caches) were modeled. RDGC can be enhanced through including more advanced architectural specifications, i.e., adaptive replacement policies, advanced indexing functions, fine-grained access bypassing, warp scheduling algorithms. Further, RDGC can be adapted for modeling cache performance in multiple simultaneous kernel execution scenarios. In addition, RDGC can benefit from introducing more realistic latency models and inclusion of atomic instructions and cache coherency protocols.

REFERENCES

[1] Patterson, D.: The Top 10 Innovations in the New NVIDIA Fermi Architecture, and the Top 3 Next Challenges. NVIDIA Whitepaper, Vol. 47, 2009.

[2] Jang, B.—Schaa, D.—Mistry, P.—Kaeli, D.: Exploiting Memory Access Patterns to Improve Memory Performance in Data-Parallel Architectures. IEEE Transactions on Parallel and Distributed Systems, Vol. 22, 2011, No. 1, pp. 105–118, doi: 10.1109/TPDS.2010.107.

[3] John, L. K.—Eeckhout, L.: Performance Evaluation and Benchmarking. CRC Press, 2005.

[4] Beyls, K.—D'Hollander, E. H.: Reuse Distance as a Metric for Cache Behavior. Proceedings of the IASTED Conference on Parallel and Distributed Computing and Systems, 2001, pp. 617–622.

[5] Pouchet, L.: Polybench/C: The Polyhedral Benchmark Suite. `http://www.cs.ucla.edu/~pouchet/software/polybench`, 2012.

[6] Che, S.—Boyer, M.—Meng, J.—Tarjan, D.—Sheaffer, J. W.—Lee, S.-H.—Skadron, K.: Rodinia: A Benchmark Suite for Heterogeneous Computing. 2009 IEEE International Symposium on Workload Characterization (IISWC 2009), 2009, pp. 44–54, doi: 10.1109/IISWC.2009.5306797.

[7] Tang, T.—Yang, X.—Lin, Y.: Cache Miss Analysis for GPU Programs Based on Stack Distance Profile. Proceedings of the 2011 31st International Conference on Distributed Computing Systems (ICDCS '11), IEEE, 2011, pp. 623–634, doi: 10.1109/ICDCS.2011.16.

[8] Nugteren, C.—van den Braak, G.-J.—Corporaal, H.—Bal, H.: A Detailed GPU Cache Model Based on Reuse Distance Theory. 2014 IEEE 20th International Symposium on High Performance Computer Architecture (HPCA), IEEE, 2014, pp. 37–48, doi: 10.1109/HPCA.2014.6835955.

[9] Wu, M.-J.—Zhao, M.—Yeung, D.: Studying Multicore Processor Scaling via Reuse Distance Analysis. ACM SIGARCH Computer Architecture News – ICSA '13, ACM, Vol. 41, 2013, No. 3, pp. 499–510, doi: 10.1145/2508148.2485965.

[10] Cui, H.—Yi, Q.—Xue, J.—Wang, L.—Yang, Y.—Feng, X.: A Highly Parallel Reuse Distance Analysis Algorithm on GPUs. 2012 IEEE 26th International Parallel and Distributed Processing Symposium (IPDPS), IEEE, 2012, pp. 1080–1092, doi: 10.1109/IPDPS.2012.100.

[11] Lee, S.—Ro, W. W.: Parallel GPU Architecture Simulation Framework Exploiting Architectural-Level Parallelism with Timing Error Prediction. IEEE Transactions on Computers, Vol. 65, 2016, No. 4, pp. 1253–1265, doi: 10.1109/TC.2015.2444848.

[12] Wu, M.-J.—Yeung, D.: Identifying Optimal Multicore Cache Hierarchies for Loop-Based Parallel Programs via Reuse Distance Analysis. Proceedings of the 2012 ACM SIGPLAN Workshop on Memory Systems Performance and Correctness (MSPC '12), ACM, 2012, pp. 2–11, doi: 10.1145/2247684.2247687.

[13] Badamo, M.—Casarona, J.—Zhao, M.—Yeung, D.: Identifying Power-Efficient Multicore Cache Hierarchies via Reuse Distance Analysis. ACM Transactions on Computer Systems (TOCS), Vol. 34, 2016, No. 1, pp. 3–30, doi: 10.1145/2851503.

[14] Bakhoda, A.—Yuan, G. L.—Fung, W. W. L.—Wong, H.—Aamodt, T. M.: Analyzing CUDA Workloads Using a Detailed GPU Simulator. 2009 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS 2009), 2009, pp. 163–174, doi: 10.1109/ISPASS.2009.4919648.

[15] Ubal, R.—Jang, B.—Mistry, P.—Schaa, D.—Kaeli, D.: Multi2Sim: A Simulation Framework for CPU-GPU Computing. Proceedings of the 21st International Conference on Parallel Architectures and Compilation Techniques (PACT '12), ACM, 2012, pp. 335–344, doi: 10.1145/2370816.2370865.

[16] Hong, S.—Kim, H.: An Analytical Model for a GPU Architecture with Memory-Level and Thread-Level Parallelism Awareness. ACM SIGARCH Computer Architecture News, ACM, Vol. 37, 2009, No. 3, pp. 152–163, doi: 10.1145/1555815.1555775.

[17] Sim, J.—Dasgupta, A.—Kim, H.—Vuduc, R.: A Performance Analysis Framework for Identifying Potential Benefits in GPGPU Applications. ACM SIGPLAN Notices – PPOPP '12, ACM, Vol. 47, 2012, No. 8, pp. 11–22, doi: 10.1145/2370036.2145819.

[18] Baghsorkhi, S. S.—Gelado, I.—Delahaye, M.—Hwu, W. W.: Efficient Performance Evaluation of Memory Hierarchy for Highly Multithreaded Graphics Processors. ACM SIGPLAN Notices – PPOPP '12, ACM, Vol. 47, 2012, No. 8, pp. 23–34, doi: 10.1145/2370036.2145820.

[19] Huang, J.-C.—Lee, J. H.—Kim, H.—Lee, H.-H. S.: GPUMech: GPU Performance Modeling Technique Based on Interval Analysis. Proceedings of the 47th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO-47), IEEE Computer Society, 2014, pp. 268–279, doi: 10.1109/MICRO.2014.59.

[20] Jia, H.—Zhang, Y.—Long, G.—Xu, J.—Yan, S.—Li, Y.: GPURoofline: A Model for Guiding Performance Optimizations on GPUs. In: Kaklamanis, C., Papatheodorou, T., Spirakis, P. G. (Eds.): Euro-Par 2012 Parallel Processing. Springer, Berlin, Heidelberg, Lecture Notes in Computer Science, Vol. 7484, 2012, pp. 920–932, doi: 10.1007/978-3-642-32820-6_90.

[21] Dao, T. T.—Kim, J.—Seo, S.—Egger, B.—Lee, J.: A Performance Model for GPUs with Caches. IEEE Transactions on Parallel and Distributed Systems, Vol. 26, 2015, No. 7, pp. 1800–1813, doi: 10.1109/TPDS.2014.2333526.

[22] Kiani, M.—Rajabzadeh, A.: VLAG: A Very Fast Locality Approximation Model for GPU Kernels with Regular Access Patterns. 2017 7th International Conference on Computer and Knowledge Engineering (ICCKE 2017), October 26–27, 2017, Ferdowsi University of Mashhad, IEEE, 2017, pp. 260–265, doi: 10.1109/ICCKE.2017.8167887.

[23] Ding, C.—Chilimbi, T.: A Composable Model for Analyzing Locality of Multi-Threaded Programs. Technical Report MSR-TR-2009-107, Microsoft Research, 2009.

[24] Jiang, Y.—Zhang, E. Z.—Tian, K.—Shen, X.: Is Reuse Distance Applicable to Data Locality Analysis on Chip Multiprocessors? In: Gupta, R. (Ed.): Compiler Construction (CC 2010). Springer, Berlin, Heidelberg, Lecture Notes in Computer Science, Vol. 6011, 2010, pp. 264–282, doi: 10.1007/978-3-642-11970-5_15.

[25] Schuff, D. L.—Parsons, B. S.—Pai, V. S.: Multicore-Aware Reuse Distance Analysis. 2010 IEEE International Symposium on Parallel and Distributed Processing, Workshops and Ph.D. Forum (IPDPSW), IEEE, 2010, pp. 1–8, doi: 10.1109/IPDPSW.2010.5470780.

[26] Wu, M.-J.—Yeung, D.: Coherent Profiles: Enabling Efficient Reuse Distance Analysis of Multicore Scaling for Loop-Based Parallel Programs. 2011 International Conference on Parallel Architectures and Compilation Techniques (PACT '11), IEEE, 2011, pp. 264–275, doi: 10.1109/PACT.2011.58.

[27] Schuff, D. L.—Kulkarni, M.—Pai, V. S.: Accelerating Multicore Reuse Distance Analysis with Sampling and Parallelization. Proceedings of the 19th International Conference on Parallel Architectures and Compilation Techniques (PACT '10), IEEE, 2010, pp. 53–63, doi: 10.1145/1854273.1854286.

[28] Wang, D.—Xiao, W.: A Reuse Distance Based Performance Analysis on GPU L1 Data Cache. 2016 IEEE 35th International Performance Computing and Communications Conference (IPCCC), IEEE, 2016, pp. 1–8, doi: 10.1109/PCCC.2016.7820638.

[29] Mittal, S.: A Survey of Techniques for Managing and Leveraging Caches in GPUs. Journal of Circuits, Systems and Computers, Vol. 23, 2014, No. 8, Art. No. 1430002, doi: 10.1142/S0218126614300025.

[30] Rogers, T. G.—O'Connor, M.—Aamodt, T. M.: Cache-Conscious Wavefront Scheduling. Proceedings of the 2012 45th Annual IEEE/ACM International Symposium on Microarchitecture, IEEE, 2012, pp. 72–83, doi: 10.1109/MICRO.2012.16.

[31] Jog, A.—Kayiran, O.—Mishra, A. K.—Kandemir, M. T.—Mutlu, O.—Iyer, R.—Das, C. R.: Orchestrated Scheduling and Prefetching for GPGPUs. ACM SIGARCH Computer Architecture News – ICSA '13, ACM, Vol. 41, 2013, No. 3, pp. 332–343, doi: 10.1145/2508148.2485951.

[32] Xie, X.—Liang, Y.—Sun, G.—Chen, D.: An Efficient Compiler Framework for Cache Bypassing on GPUs. 2013 IEEE/ACM International Conference on Computer-Aided Design (ICCAD '13), IEEE, 2013, pp. 516–523, doi: 10.1109/ICCAD.2013.6691165.

[33] Kim, K. Y.—Baek, W.: Quantifying the Performance and Energy Efficiency of Advanced Cache Indexing for GPGPU Computing. Microprocessors and Microsystems, Vol. 43, 2016, pp. 81–94, doi: 10.1016/j.micpro.2016.01.003.

[34] CUDA NVIDIA. C Programming Guide Version 4.0. NVIDIA Corporation, 2011.

[35] NVIDIA: Maxwell Tuning Guide, 2017. Available at: `http://docs.nvidia.com/cuda/maxwell-tuning-guide/index.html`, Accessed 18-February-2017.

[36] Li, C.—Song, S. L.—Dai, H.—Sidelnik, A.—Hari, S. K. S.—Zhou, H.: Locality-Driven Dynamic GPU Cache Bypassing. Proceedings of the 29th ACM International Conference on Supercomputing (ICS '15), ACM, 2015, pp. 67–77, doi: 10.1145/2751205.2751237.

[37] Mei, X.—Chu, X.: Dissecting GPU Memory Hierarchy Through Microbenchmarking. IEEE Transactions on Parallel and Distributed Systems, Vol. 28, 2017, No. 1, pp. 72–86, doi: 10.1109/TPDS.2016.2549523.

Mohsen Kiani is currently a Ph.D. student in computer engineering at Razi University, Kermanshah, Iran. His main research interests include computer architecture, many-core architectures, GPGPU, and performance modeling and analysis.

Amir Rajabzadeh received his B.Sc. degree in telecommunication engineering from Tehran University, Iran, in 1990 and his M.Sc. and Ph.D. degrees in computer engineering from Sharif University of Technology, Iran, in 1999 and 2005, respectively. He was a visiting researcher in the Embedded Systems Laboratory, University of Leicester, UK in summer 2005 and in the CARG Group, Ottawa University, Canada in 2012–2013. He has been working as Assistant Professor of computer engineering at Razi University, Kermanshah, Iran since 2005. He was the Head of the Computer Engineering Department (2005–2008) and the Education and Research Director of the Engineering Faculty (2008–2010) at Razi University. He has authored several journal papers and other refereed publications. His main areas of interests are computer architecture, high performance computing and fault-tolerant systems design. He has earned one world, six international, and five national awards in robotic competition, and one national award in mobile computing.

Computing and Informatics, Vol. 38, 2019, 454–472, doi: 10.31577/cai_2019_2_454

DEVELOPING AND EVALUATING ECM DATA PERSISTENCE ARCHITECTURE

Juris Rats

RIX Technologies
Blaumana 5a-3
Riga, LV-1011, Latvia
e-mail: juris.rats@rixtech.lv

Abstract. A conceptual data persistence architecture and methodology to evaluate its performance is created. Results of the empirical research indicate that the architecture created is convenient for high intensity processing of large ECM data volumes. The synthesis of SQL and NoSQL technology allows to handle high volume transactions on current data and full-text search on large sets of history data. The history data is moved from SQL to NoSQL data store thus allowing to use a cluster of commodity hardware to store major volume of ECM data. Scaling out (increasing a number of cluster nodes) is less expensive than scaling up (buying a more powerful server hardware) that is normally needed to upgrade SQL database.

Keywords: Polyglot persistence, NoSQL, ECM, Elasticsearch, hot-warm architecture

Mathematics Subject Classification 2010: 68M14, 68U35

1 INTRODUCTION

Forrester research [10] shows 40 % of firms were implementing and expanding big data technology adoption. Another 30 % were planning to adopt big data in the next 12 months. 25 % annual growth for NoSQL technologies is predicted.

Large number of technologies (Google BigTable, Apache Hadoop, NoSQL databases of all types, etc.) evolved to address big data. They propel a number of important paradigm shifts; the most important being:

- understanding the importance of clustered solutions;
- understanding there is no "one best choice" for all cases [19];
- understanding it might be not enough to employ one data persistence technology for the use case.

The third shift caused the introduction of the polyglot persistence – a process for storing data in the best database available, no matter the data model and data storage technology [9].

ECM (Enterprise Content Management) solutions deal with a large and fast expanding amounts of data and user requests. Traditionally SQL databases are used to persist ECM data. The SQL solutions do not scale out well, but it is a general opinion that NoSQL databases are not good on transaction support hence they do not qualify.

The aim of the research is to show that ECM data can be split into two parts (current and history data) that each are manipulated differently. The current data is one that is created or updated and need transaction support. The history data is not updated but searched, aggregated or retrieved instead. The history data accounts for the major part of the data volume hence clustered NoSQL database might be the best fit for this part of data.

The MS SQL database is used to store and manipulate current data while Elasticsearch (ES) [3, 13] – to store and query current and history data. Current data is replicated from MS SQL to ES as it is changed while history data is removed from MS SQL and stored in ES exclusively.

The research aims as well to create a methodology and a tool for performance measurement of a data persistence solution on a representative ECM data and request flows consisting of randomly dispersed sequences of user requests. The methodology is based on results of the earlier research [17].

Section 2 outlines the related work. The typical workflow scenario, objects involved and data statistics are described in Section 3, Section 4 gives a brief description of the technology used, while Section 5 illustrates the architecture proposed by the research. They are followed by Section 6 describing test data, Section 7 describing the methodology used to evaluate the performance of the proposed architecture, and Section 8 analyzing the results of the evaluation. Section 9 outlines the conclusions and is followed by acknowledgements.

2 RELATED WORK

The term NoSQL (Not only SQL) was initially used by Carlo Strozzi [11] in 1998. More than 225 NoSQL platforms of various kinds have been developed since [8]. The new platforms generally are aimed at specific problem areas and there are no general suggestions on what solution would be the best for a specific use case.

Polyglot persistence paradigm has been researched by various authors. A contemporary outline is given by Sadalage and Fowler [18]. They write that using

a single database engine for all of the requirements usually leads to non-performant solutions; storing transactional data, caching session information, traversing graph of customers and the products their friends bought are essentially different problems [18]. Use of several technologies should be considered instead of sticking to one. Document store is the most convenient NoSQL technology for ECM [16], still the polyglot persistence approach would be the best fit here to use strong transaction support of relational database for data maintenance and NoSQL document store for fast information search and retrieval. Some solutions already use a similar approach, e.g. using MySQL database together with Elasticsearch [21].

Clustered solutions allow to cope with large and rapidly expanding data and request volumes, but they are inherently more complicated than their one server counterparts. The complexity relates, in particular, to the increased number of parameters to watch and configure to tune the solution as well to the methods of performance measurement.

To develop a performance measurement tool for the performance evaluation of the data persistence architecture proposed, the number of benchmarking tools have been reviewed, e.g. YCSB [1], BigBench [12], GridMix [14], PigMix [6], HBench:java [20] and GraySort [7]. GridMix, PigMix, HBench:java and GraySort are dedicated for benchmarking of specific systems. BigBench is a benchmark proposal dedicated for a specific business model (product retailer) and specific two layer data model with ETL backed data transfer between layers.

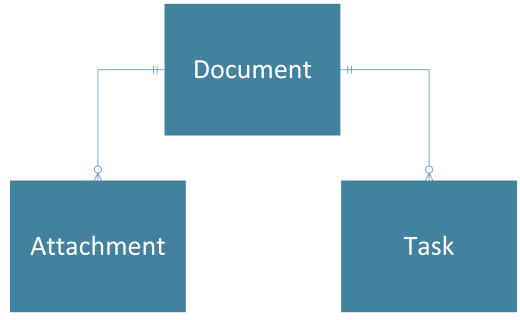


Figure 1. Data model

YCSB (Yahoo! Cloud Service Benchmark) is a generic framework and tool used for benchmarking of a number of distributed systems (Cassandra, HBase, MongoDB, Elasticsearch, Redis etc.). The shortcomings are as follows:

- It is assumed (see e.g. [5, 12]) that a request flow (workload) consists of mutually independent data requests; in reality user requests are grouped in a sequences of dependent requests, the model should be expanded to cover this.
- All test data ([5, 12]), including the texts used for full-text search, is generated – the generated text does not represent real data well when measuring search request performance.

3 ECM DATA ANALYSIS

The research is focused on a document management, but the patterns are the same for web content management, records management and workflow management as well. Figure 1 shows the data model used in our research. Table 1 gives the detailed description of the data attributes.

	D	A	T	Comments
docNum	x	x	x	Document number. Replicated from the document object to all child attachment and task objects.
inCharge	x	x	x	Person in charge of the document (Document and child Attachments) or for the task (Task object).
author			x	Creator of the task.
canRead	x	x		List of user ids having access to the document and its attachments because they are authors or persons in charge of some child task of the document. Replicated to the child attachment objects.
case	x	x		Case the document is attached to. Replicated.
docType	x			Document type.
date	x	x	x	Indexing date and time of the document. Replicated to all child attachments and tasks. Determines the ES index the document and its children are allocated to.
deadline			x	Task deadline.
folder	x	x		Folder the document is attached to. Replicated.
project	x			List of projects a document is attached to.
status	x		x	Document or task status.
summary	x	x		Document title, attachment title.
content		x		Attachment content.
name			x	Task type.
comment			x	Task comments.

Table 1. Document, attachment and task attributes

The document here is a placeholder for zero or more attachment files, that an organization receives (or sends) in one go. The task is an action that an employee has to perform on the document to move it along the workflow. Tasks are of several types and multiple tasks may be related to a document.

One of regular scenarios is – an organization receives e-mail with one or several attachments from a partner or a customer (the document), a new document is created, the document and attachments are indexed, a workflow of tasks (assigned to one or more employees) is manually or automatically attached. Employees complete tasks (and/or assign new ones), create a response document (with one or several attachments) that is sent back to the correspondent. Described data items form the major part of data volume for document management process.

The analysis shows that users mostly interact with the current data and the rest is accessed scarcely. Figure 2 shows this dependency of user activity volume from the document age for 5 customer databases (K1–K5). The figure shows percentage of data requests (mixed read, write, search, aggregate) addressed to data of each document age group (up to 1 month, 1 to 2 months, etc.).

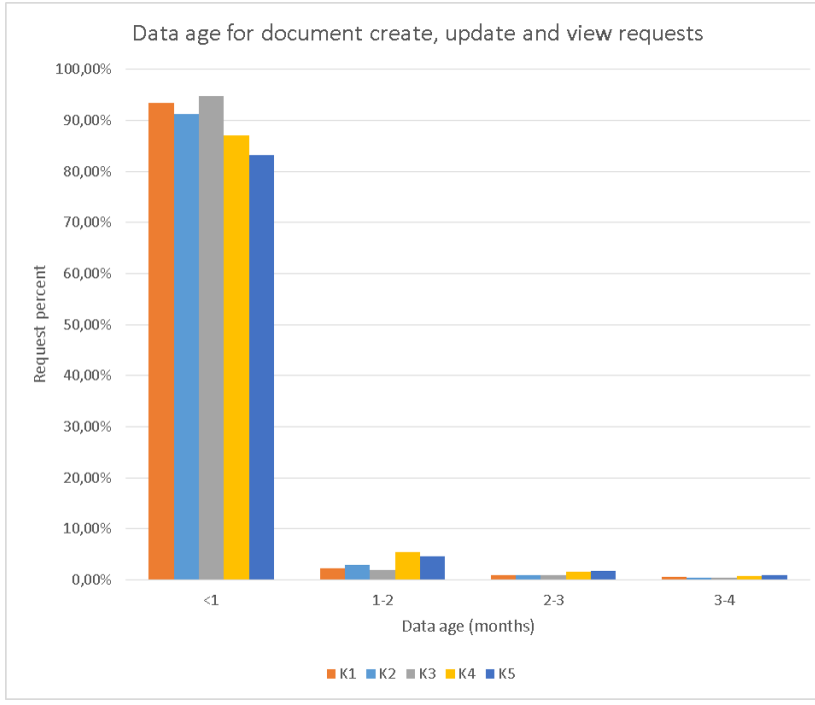


Figure 2. Data age for document create, update and view requests

To make use of the fact that most of the data is accessed scarcely we should build our persistence model in a way that allows separate storage and processing of frequently and scarcely used data. We propose to create two data stores:

- Current data store for create, update and delete requests;
- General data store for search, aggregation and retrieval.

This model would allow to use separate persistence technologies for transaction support (Current data store) and for search and aggregation in a large, expanding data volumes (General data store).

4 ELASTICSEARCH

Elasticsearch (ES [3, 13]) is a search engine built on top of Apache Lucene [15] – perhaps the most advanced, high-performance, and fully featured search engine library in existence today. Elasticsearch uses Lucene internally for indexing and searching, but it aims to make full-text search easy by hiding the complexities of Lucene. Elasticsearch is used for our research because of its advanced text searching features and because it is:

- a distributed real-time document store where every field is indexed and searchable;
- a real-time analytics engine;
- capable of easy scaling to hundreds of servers and petabytes of structured and unstructured data.

Unlike the relational databases ES stores all the data into indexes. Indexes store both the data and all the information necessary for search. ES index does not have a predefined structure, but it can be configured (through field mappings) according to the types of search anticipated for the particular fields. One might define if the particular field is to be searched or not, should it be ready for full-text search, ranged search, exact match, geographic search, etc.

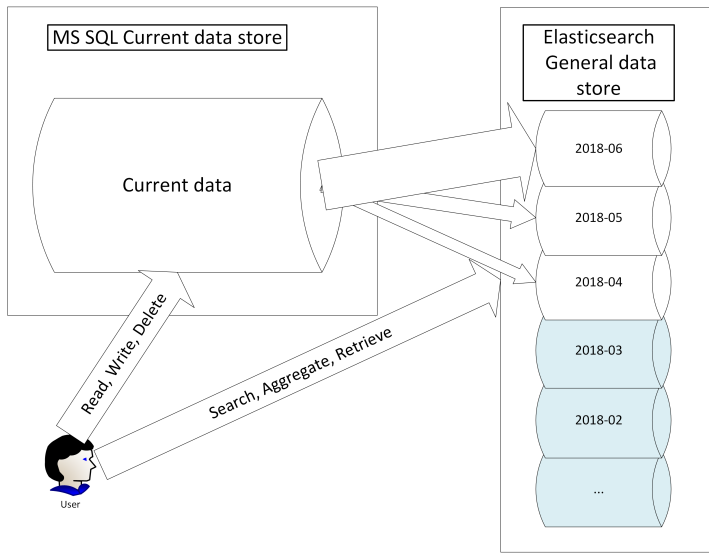


Figure 3. Proposed architecture

5 THE PERSISTENCE ARCHITECTURE

Figure 3 shows the proposed architecture. General data store consists here of a time dependent ES indexes. Time-dependent means here that an index contains data related to a particular time period (e.g. a month or a year). This allows to store data for different time periods on different nodes in a distributed infrastructure.

The following features are shown there:

- user creates, updates and deletes data in the Current data store;
- changes in the Current data store are replicated to the General data store;
- user searches, aggregates and retrieves data from the General data store;
- data in the General data store are split into time dependent indexes;
- as time passes indexes are switched to read-only mode (indexes highlighted in blue);
- as long as indexes of the General data store are switched to read-only mode, the data of the matching time periods may be removed from the Current data store.

Indexes in blue thus differ from the white ones because they are read-only and they may have no backing data in the Current data store. When implementing the architecture customer may decide when to make time periods read-only and when to remove data from the Current data store. Customer may decide as well to keep all the history time periods online, or put them offline at some point. The latter option allows to reduce infrastructure costs at an expense of increased latency for some rare user requests.

5.1 Hot-Warm Architecture Model

As concerns the General data store, this research follows the hot-warm architecture paradigm [2] for separate handling of frequently and scarcely used data. The hot-warm paradigm suggests to use different groups of cluster nodes to store frequently used (hot) data and scarcely used (warm) data. In respect to our data model new time periods are allocated to hot nodes and older time periods are switched to warm nodes when appropriate. ES allows to do this easily and transparently from the application. One has to issue a simple request that changes appropriate attribute of the index and ES automatically moves the index from hot nodes to some warm nodes. It is important to stress that moving data from hot to warm nodes concerns only the General data store, and it is independent from the process of moving data from the Current to General data store.

Figure 4 shows hot-warm ES cluster with 2 hot nodes and 3 warm nodes. Cluster has as well 3 master eligible nodes. These nodes elect a single master node of the cluster that is responsible for the cluster management functionality. Other two are here to replace master in case of emergency. 3 master eligible nodes and two of them available is the minimal configuration for the healthy cluster as this prevents

so called split brain scenario [2] when more than one node is elected (because of the lost communication) as a master. This situation is dangerous as two masters act on a cluster data concurrently that may lead to data corruption or loss.

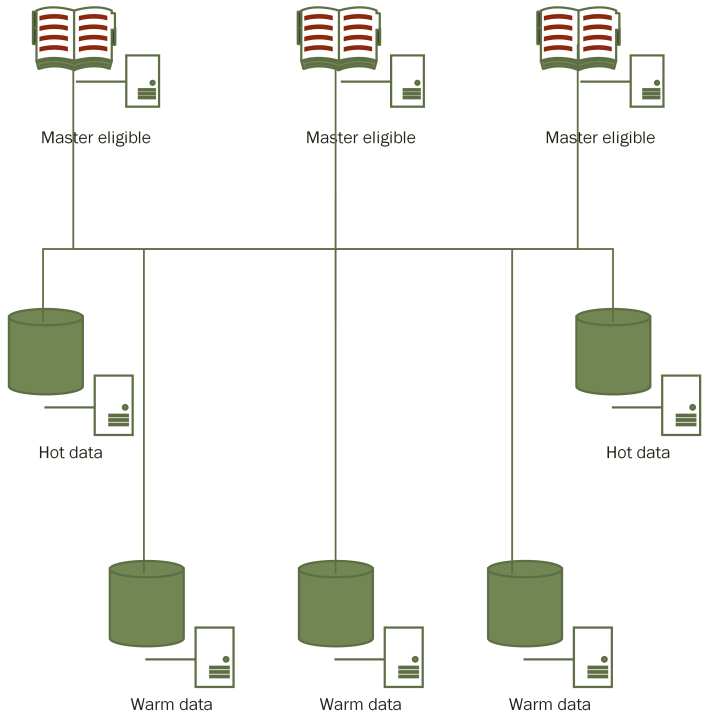


Figure 4. Hot-warm Elasticsearch cluster

With the architecture in place we will discuss below its advantages.

5.2 No Locking

ES uses Lucene [15] immutable indexes thus there is no need to lock index when writing data [3]. New index segments are created to index new data instead while index segments are merged in background later on. Thus General data store is available for search and data retrieval no matter how intense is the flow of new data replicated from the Current data store.

Downside of the immutable index technology is that newly indexed data does not become available for search instantly but with a delay instead. The delay varies from below a second normally to tens of seconds or more when system is heavy loaded. In contrary to the relational database case this does not result though in all search requests waiting while locks are released. Latest updates might not be included in the search results instead. ES provides several options to deal with this

problem – application may return the control to user not waiting for index refresh (to proceed with his work) or waiting for index to be refreshed (if user wants to see his changes before to proceed).

5.3 Scale Out

ES index consists of number of shards. When the data volume grows new nodes can be added to the cluster. ES automatically relocates shards when new nodes added. Thus ES index with e.g. 5 shards can run on 1 to 5 node cluster.

Replicas allow to scale out ES database even more. Primary shard and its replicas are allocated each to a different cluster node to achieve this. ES index with 5 shards and one replica can run on 2 to 10 node cluster.

Parameter	Comments
Time phased indexes	Major part of user requests is directed to the current data, that way major part of the all requests may be addressed to a small part of all indexes; this makes the request lighter and decreases the response time.
Read-only indexes	ES provides for optimization of read-only indexes; normally ES index shard consists of multiple (several tens to hundreds) segments, every search request is executed against all the segments; if index is not supposed to be changed anymore, segments can be merged into one; this speeds up search requests.
Hot-warm architecture	Having separate groups of cluster nodes for current (hot) and history (warm) data allows to deploy more powerful hardware for hot nodes to support low search latency (and save money on hardware for warm nodes).

Table 2. Means to improve search speed

5.4 Availability and Fast Search

Replicas are redundant data copies. Thus in addition to decreased search request latency they provide increased data availability. Index shard is available if the primary shard or one of replicas is available.

Elasticsearch has proven to be one of the fastest and richest search engines capable of handling very large data and user request volumes. The proposed architecture provides a couple of means to profit from these ES advantages (Table 2).

6 TEST DATA

The test data stores are created following the methodology presented in [17], elaborated and tuned for the current use case. Values for all but the fields employed

for the full text search are generated according to the frequencies calculated out of the real data in five customer databases (see Section 3). Texts for the two fields used for performance measurement of full-text search (attachment content and task comments) are extracted from Common Crawl – the open repository of web crawl data [4]. The data set of September 2017 is used that contains extracted texts from more than 2 billion pages in English or similar (Latin based charset) languages. This enables for creating of large data sets usable for full-text search.

ES hot-warm cluster of 6 nodes is created in MS Azure cloud and three large volume test data stores are generated (using statistics of the customer databases mentioned in Section 3) for performance tests (Table 3). Store names indicate data model and time span used. Shared model uses shared index for all three data objects (documents, attachments, tasks) while multi model stores each object in a separate index (see Section 4 for ES storage explanation). Yearly model uses index for a year of data while monthly – for a month of data.

Store	Time Span	Index Shards	Object Count (millions)	Volume (GB)
Shared-year	6 years 2 months	35	1140	1 200
Shared-month	14 months	14	145	160
Multi-month	14 months	42	145	160

Table 3. Test databases

7 METHODOLOGY FOR PERFORMANCE EVALUATION

The aim of the evaluation is to measure how the solution performs:

- on large volumes of data specific for the use case in question;
- on a user request flow specific for the use case in question;
- on a specific cluster configuration one has or would like to change.

The methodology developed in our earlier research [17] is expanded to:

- support time partitioned data model to allow separate management of current and history data;
- support specific user request flows for current and history data;
- comprise a number of cluster configuration parameters.

Below these three dimensions of the proposed methodology are described in more detail.

7.1 Time Partitioned Data Model

The data storage is split into time specific ES indexes (see Section 4). The yearly and monthly indexes are analyzed for comparison. The date field of the document

determines an index where the document and all its child attachments and tasks are routed to.

To ensure realistic search results test database is generated according to the following principles:

- Document, attachment and task metadata are generated in accordance with the frequencies typical for the real databases (based on statistics of 5 customer databases, described in Section 3).
- Attachment content and task comment data is extracted from the commoncrawl.org repository that contains texts of the real web pages; these fields are used for full-text search.

7.2 User Request Flow

The performance measurements are based on a user interaction flow (interaction schedule) model developed in our earlier research [17]. The user interaction flow is decomposed into user business activities (like – show my urgent tasks). The business activities are represented as sequences of user interactions (i.e., user request that can be executed by one or more data requests without user intervention) as shown in Figure 5 in a portion of a sample user interaction flow specification. User interactions are further decomposed as series of data requests not shown in a specification sample.

```
{
    "sequences" : [
        { "seq" : ["aD"], "count" : 100},
        { "seq" : ["aD","aA"], "count" : 400},
        { "seq" : ["aD","aA","aT"], "count" : 100},
        ...
        { "seq" : ["sFPa","gA"], "count" : 5}
    ],
    "tasks" : {
        "aD" : { "archi" : false, "pars" : ["addDoc"],"tf" : 0},
        "cD" : { "archi" : false,  "pars" : ["changeDoc"],"tf" : 0},
        ...
        "gT" : { "archi" : false,  "pars" : ["getTask"],"tf" : 0}
    },
    "tf" : [
        {"freq" : 90, "threshold1" : 1000, "threshold2" : 5000},
        {"freq" : 90, "threshold1" : 2000, "threshold2" : 8000},
        {"freq" : 80, "threshold1" : 5000, "threshold2" : 15000}
    ]
}
```

Figure 5. Sample user request flow description

The specification describes sequences (user business activities), tasks (user interactions and data requests) and time functions (tf). A sequence description:

- lists user interactions (tasks) of the sequence, e.g., a sequence consisting of two user interactions – aD (add a document) and aA (add an attachment),

- specifies an average times (per year) an average user would run the sequence (user business activity).

A task (user interaction) description specifies if the interaction is addressed to current data ("archi": false), specifies the user interaction name (addDoc) and optional parameters, and specifies the time function used for performance measurements.

A time function description (tf) specifies what latency (in milliseconds) has to be assured for what percentage of the user interaction (e.g. 1 second for 90 % of interactions, 5 seconds for all the interactions). Figure 6 shows the first 30 seconds of the sample evenly distributed user interaction schedule for 8000 users.

To evaluate performance a set of user interaction sequences is used that includes search (e.g., full-text search inside document content), filtering and processing of aggregates, as well as document, attachment and task creation and modification. The results of the research [17] are used to assume frequencies of execution of the user interaction sequences by an ECM user. The list of user interaction sequences includes the requests to the current as well to the history data. User interaction simulation model is tuned to the time dependent index structure of our architecture model allowing to explicitly direct a part of requests to a particular index (or several indexes).

7.3 Cluster Parameters

Measuring a performance of a clustered solution is a challenge because we have to assess both performance of a healthy cluster and emergency performance (recovery from node unavailability). On a production cluster we can measure mainly a performance of a healthy cluster (as it mainly is healthy). Therefore we need a sibling cluster to play with crashes and recoveries. This is a costly option hence a cloud infrastructure should be considered as the sibling cluster might be created there when needed and disposed afterwards.

The performance tests are executed on a number of configurations. A number of parameters analyzed are described below.

Data model. Elasticsearch nested objects and parent-child constructs are not used as they are resource hungry. Relationship between documents and child attachments and tasks are maintained at an application level. Two different data models are evaluated – all three types of objects stored in a single ES index (shared model), and each object type stored in separate index (multi model). The shared model needs 3 times less indexes while the multi model takes less storage space.

Time span for a single index. Yearly and monthly index is tested. The first one needs less indexes, the second one allows more control over data (e.g. indexes may be moved from hot to warm nodes on a monthly basis).

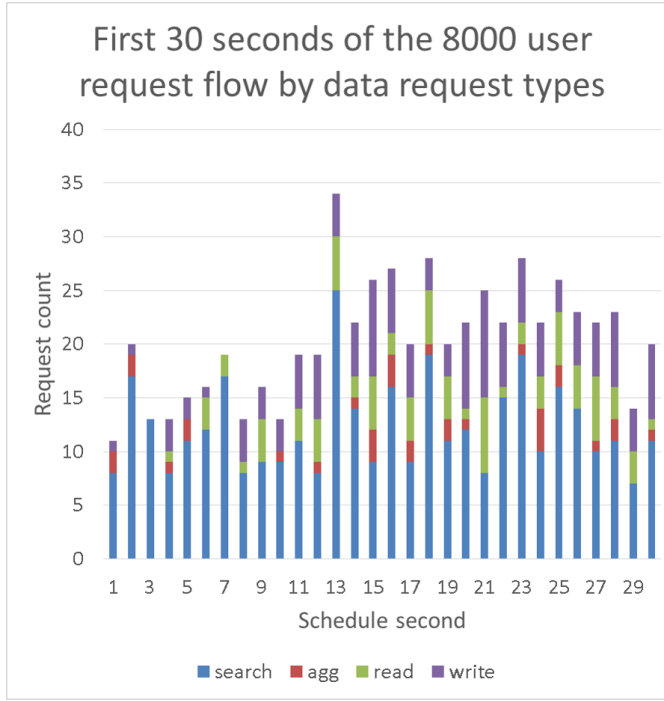


Figure 6. Sample request flow by request types

Cluster configuration and node count. The impact of overall node count, as well as of several role configurations (e.g. dedicated master eligible nodes versus warm nodes configured as master eligible nodes) are assessed.

Number of replicas. Cluster performance is evaluated for 1-3 replicas for an index shard.

Node hardware characteristics. RAM and heap volume as well as disk volume and speed impact the overall cluster performance. Several configurations of these parameters are evaluated.

Overall data volume. The performance of the cluster for data object volumes worth of 3 to 6 years of data is evaluated. The yearly amount of data is 190 million of data objects (documents, attachments and tasks).

User interaction flow characteristics. Measurements are performed for user interaction flows of different volumes (up to average 27 user interactions per second). User interaction flows are generated to contain mixed sequence of search, aggregation, retrieval, as well as data insert and update requests. User interaction flows are generated for 13 minute test runs (3 minutes warming and 10 minute rating phase) and contain 230 to 18 870 data requests each.

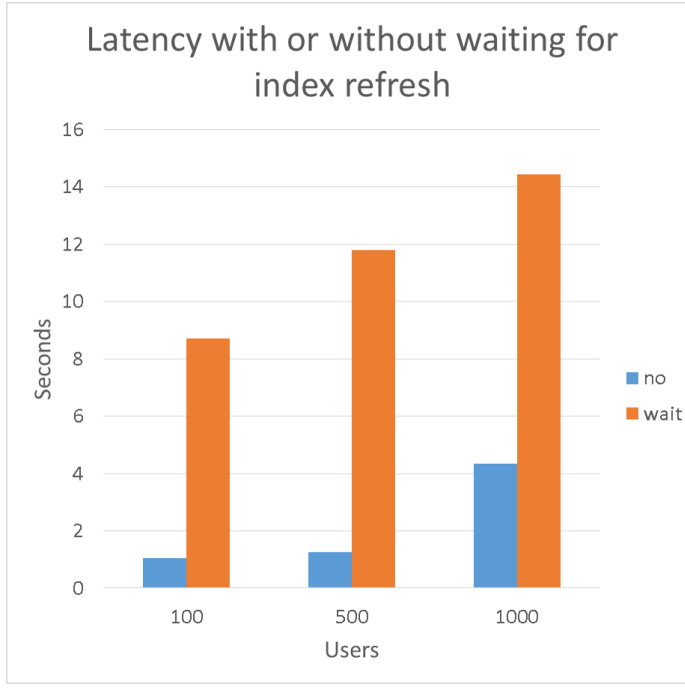


Figure 7. Latency with or without waiting for index refresh

8 RESULTS OF THE MEASUREMENTS

240 tests are executed to measure different configurations of above described parameters. One test here is an execution of a 13 minute (3 minute warming and 10 minute rating phase) long request flow.

Generally the analysis of the results supports the opinion dominant in ES support forums and elsewhere that cluster and data model parameters must be tuned for a particular use case. A number of interesting patterns have been observed and they are explained in the Sections 8.1, 8.2 and 8.3.

8.1 RAM and Index Volume Ratio

ES node loads index data from disk to RAM when started. The volume of data to be loaded in RAM depends on the total index data volume and on field mappings (see Section 4). A number of configurable ES parameters may influence this (e.g. types of indexing mapped for particular fields). If RAM volume of the node is too small to load all the necessary data, it is not possible to start the node. The RAM and index volume ratio (to be loaded on node start-up) thus must be carefully watched by the administrator. Our tests show that at least 7 GB RAM is necessary for a node to

handle 1.2 TB of disk index volume for our index mappings. To be on the safe side it is better to have excess amount of RAM though.

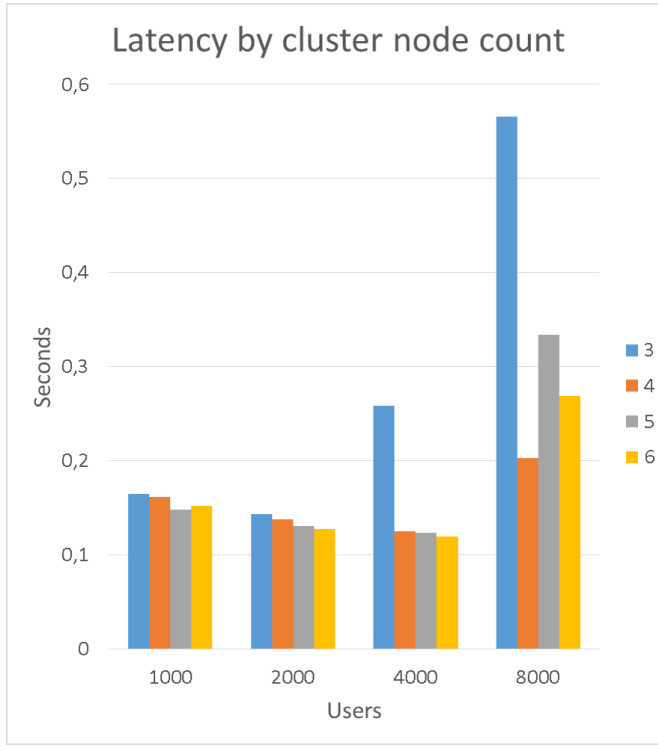


Figure 8. Latency of 3-6 node cluster for 7 GB node RAM

8.2 Index Refresh

ES by default refreshes indexes every second. This means that every second ES takes the index segments with freshly indexed data and makes them available for search (further indexing requests go to newly created segments). New data gets available to search though only when this index refresh process is complete. When cluster load grows index refresh process may slow down considerably to take several tenths of seconds or more. Figure 7 shows the pattern for shared-year data store (1.14 billion objects), cluster with 5 data nodes, 1 replica, 7 GB RAM, HDD disks.

Index requests must be implemented to wait for index refresh only when it is absolutely necessary, e.g. if user needs to immediately see the new/changed data. Index requests that do not wait for index refresh perform better when cluster load grows.

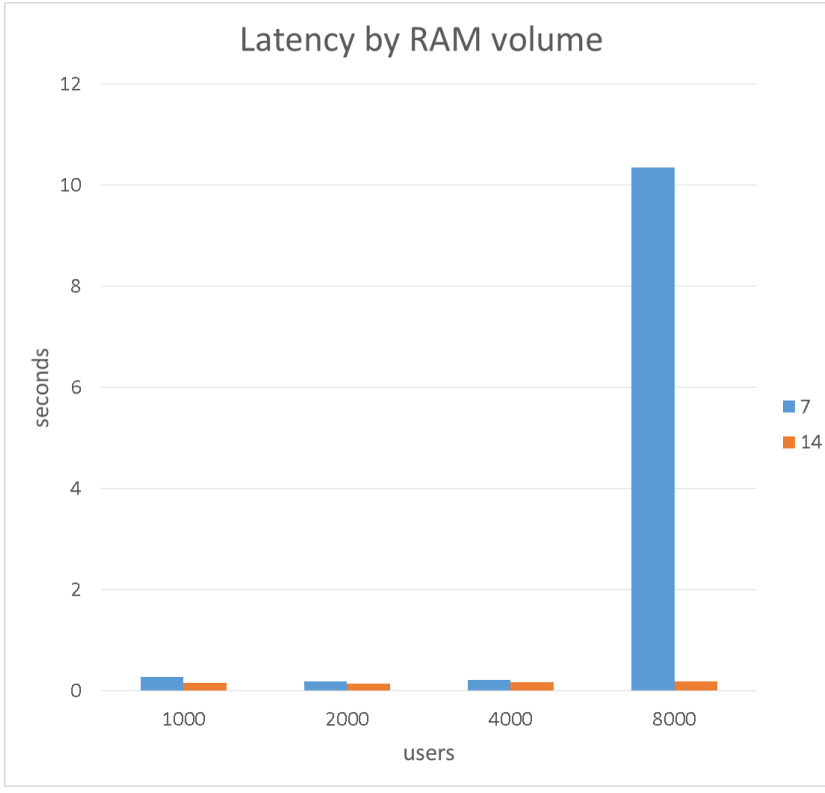


Figure 9. Latency by node RAM volume for cluster with 3 nodes

8.3 Scale Up or Scale Out

Scaling up or out are two options that frequently are compared. The performance tests executed show that ES cluster node performs better when it has plenty of RAM volume. Nodes with 7 GB and 14 GB RAM are tested and it appears (see Figures 8 and 9) that for large data request amounts it is better to expand RAM from 7 GB to 14 GB than to double the cluster node count.

For larger RAM volumes it might be more convenient to increase node count as smaller nodes mean less damage and shorter recovery times when a node gets down.

8.4 Cloud Infrastructure Costs

The performance tests show what could be the potential infrastructure costs for a data store and user request volume we are interested in (see Table 4).

Data volume (years)	6 years 2 months
Data volume (million objects)	Documents – 120, attachments – 120, tasks – 900
Data request flow	Up to 27 per second
Hot node specifications	14 GB RAM, SSD 1 TB
Warm node specifications	14 GB RAM, HDD 2 TB
MS Azure cloud cost per month (EUR)	2 090

Table 4. MS Azure cloud cluster configuration and costs

The data and request volumes correspond to the ones of comparatively large organization (8 000 users, 1.3 million documents per month).

9 CONCLUSIONS

The polyglot persistence architecture is defined consisting of two data stores – the Current data store and the General data store. Data is inserted/updated into the Current data store and searched/accessed in the General data store. MS SQL is employed for the Current data store to ensure strong transaction support while Elasticsearch is used for the General data store to provide fast execution of large volumes of search requests on large volumes of data. The General data store is split into time dependent indexes and handled by a clustered ES solution of a hot-warm architecture to allow for separate management of current and history data.

The methodology and tools for performance evaluation of the proposed architecture are created. The methodology allows for creation of test data and test workloads used to measure the performance and to evaluate impact of a number of parameters (node count and configuration, replica count, RAM volume, disk speed, total data amount, user request volume, etc.). Analysis of the evaluation results uncovers a number of interesting patterns. The results on several sets of parameters may be used to evaluate and compare alternative ways of cluster scaling (e.g. more nodes against more node RAM) for a production system.

The architecture is considered to be convenient for ECM solutions of large data and user request amounts. The measurements performed show that MS Azure hosted cluster configuration costing about EUR 2 000 per month would handle database of 1.14 billion objects (documents, attachments and tasks) for average 27 per second flow of mixed data create, update, read, search and aggregate requests.

Acknowledgement

The research is co-funded by the European Regional Development Fund (ERDF) (project No. 1.2.1.1/16/A/007).

REFERENCES

[1] Abubakar, Y.—Adeyi, T. S.—Auta, I. G.: Performance Evaluation of NoSQL Systems Using YCSB in a Resource Austere Environment. International Journal of Applied Information Systems, Vol. 7, 2014, No. 8, pp. 23–27.

[2] Bennacer, S.: "Hot-Warm" Architecture in Elasticsearch 5.x. 2017. https://www.elastic.co/blog/hot-warm-architecture-in-elasticsearch-5-x.

[3] Brasetvik, A.: Elasticsearch from the Bottom Up, Part 1. 2013, available at: https://www.elastic.co/blog/found-elasticsearch-from-the-bottom-up.

[4] Common Crawl, open repository of web crawl data. Available at: http://commoncrawl.org/.

[5] Cooper, B. F.—Silberstein, A.—Tam, E.—Ramakrishnan, R.—Sears, R.: Benchmarking Cloud Serving Systems with YCSB. Proceedings of the 1st ACM Symposium on Cloud Computing (SoCC '10), 2010, pp. 143–154, doi: 10.1145/1807128.1807152. Available at: https://www2.cs.duke.edu/courses/fall13/cps296.4/838-CloudPapers/ycsb.pdf.

[6] Dai, J.: PigMix. 2013, available at: https://cwiki.apache.org/confluence/display/PIG/PigMix.

[7] Dvorský, M: History of Massive-Scale Sorting Experiments at Google. 2016, available at: https://cloud.google.com/blog/big-data/2016/02/history-of-massive-scale-sorting-experiments-at-google.

[8] Edlich, S.: NOSQL Databases. Available at: http://nosql-database.org/.

[9] Foote, K. D.: Utilizing Multiple Data Stores and Data Models: Is Polyglot Persistence Worth It? 2017, available at: http://www.dataversity.net/utilizing-multiple-data-stores-data-models-polyglot-persistence-worth.

[10] Forrester Forecasts Big Data Tech Market Will Grow ∼3x Faster Than Overall Tech Market. 2016, available at: https://go.forrester.com/press-newsroom/forrester-forecasts-big-data-tech-market-will-grow-3x-faster-than--overall-tech-market/.

[11] Fowler, A.: NoSQL for Dummies. John Wiley and Sons, Inc., Hoboken, New Jersey, 2015.

[12] Ghazal, A.—Rabl, T.—Hu, M.—Raab, F.—Poess, M.—Crolotte, A.—Jacobsen, H.-A.: BigBench: Towards an Industry Standard Benchmark for Big Data Analytics. Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data (SIGMOD '13), New York, June 2013, pp. 1197–1208, doi: 10.1145/2463676.2463712.

[13] Gormley, C.—Tong, Z.: Elasticsearch: The Definitive Guide. O'Reilly Media, Inc., Sebastopol, CA, 2015.

[14] GridMix. Available at: https://hadoop.apache.org/docs/r1.2.1/gridmix.html.

[15] McCandless, M.—Hatcher, E.—Gospodnetić, O.: Lucene in Action. 2nd Edition. Manning Publications Co., 2019. Available at: https://livebook.manning.com/#!/book/lucene-in-action-second-edition.

[16] Potts, J.: Alfresco, NOSQL, and the Future of ECM. 2010, available at: http://ecmarchitect.com/archives/2010/07/07/1176.

[17] Rats, J.: Simulating User Activities for Measuring Data Request Performance of the ECM Visualization Tasks. International Journal of Applied Mathematics and Informatics, Vol. 9, 2015, pp. 96–102.

[18] Sadalage, P. J.—Fowler, M.: NoSQL Distilled: A Brief Guide to the Emerging World of Polyglot Persistence. Pearson Education, Inc., Upper Saddle River, New Jersey, 2012.

[19] Vorhies, B.: Polyglot Persistence? 2015, available at: http://data-magnum.com/polyglot-persistence.

[20] Zhang, X.—Seltzer, M.: HBench:Java: An Application-Specific Benchmarking Framework for Java Virtual Machines. ACM 2000 Java Grande Conference, 2000. Concurrency and Computation: Practice and Experience, Vol. 13, 2001, No. 8–9, pp. 775–792, doi: 10.1002/cpe.578. Available at: https://www.eecs.harvard.edu/margo/papers/javagrande00/paper.pdf.

[21] Zwierzynski, P.: How We Use Elasticsearch. 2015, available at: https://www.bigeng.io/how-we-use-elasticsearch.

Juris Rats is Project Manager at RIX Technologies, Riga, Latvia. He is the Latvian University graduate and there he received his Ph.D. in computer science. He was working as a programmer, software and business analyst, IT project and team manager, university lecturer and researcher in several IT companies and universities in Latvia as well as in Germany and UK. His research interests include big data technologies, NoSQL databases, distributed systems and ECM systems. Lately, he was involved in four successfully completed ERDF co-financed research projects. Currently, he has been working on ERDF co-financed research project on use of hybrid data models for effective processing of enterprise data.

Computing and Informatics, Vol. 38, 2019, 473–496, doi: 10.31577/cai_2019_2_473

MINING QUERY PLANS FOR FINDING CANDIDATE QUERIES AND SUB-QUERIES FOR MATERIALIZED VIEWS IN BI SYSTEMS WITHOUT CUBE GENERATION

Atul Thakare, Srijay Deshpande
Amit Kshirsagar, Parag Deshpande

Department of Computer Science and Engineering
Visvesvaraya National Institute of Technology
South Ambazari Road, Nagpur, Maharashtra
India - 440010
e-mail: {aothakare, deshpandesrijay, amit.kshirsagar185}@gmail.com,
psdeshpande@cse.vnit.ac.in

Abstract. Materialized views are important for optimizing Business Intelligence (BI) systems when they are designed without data cubes. Selecting candidate queries from large number of queries for materialized views is a challenging task. Most of the work done in the past involves finding out frequent queries from the past workload and creating materialized views from such queries by either manually analyzing workload or using approximate string matching algorithms using query text. Most of the existing methods suggest complete queries but ignore query components such as sub queries for creation of materialized views. This paper presents a novel method to determine on which queries and query components materialized views can be created to optimize aggregate and join queries by mining database of query execution plans which are in the form of binary trees. The proposed algorithm showed significant improvement in terms of more number of optimized queries because it is using the execution plan tree of the query as a basis of selection of query to be optimized using materialized views rather than choosing query text which is used by traditional methods. For selecting a correct set of queries to be optimized using materialized views, the paper proposes efficient specialized frequent tree component mining algorithm with novel heuristics to prune search space. These frequent components are used to determine the possible set of candidate queries for creation of materialized views. Experimentation on standard, real and synthetic data sets, and also the theoretical basis, proved that the proposed method is able to optimize a large number of queries with less number of mate-

rialized views and showed a significant improvement in performance compared to traditional methods.

Keywords: Query optimization, view selection, tree mining, query plans, materialized views, query response time

Mathematics Subject Classification 2010: 68Uxx

1 INTRODUCTION

Most of the BI systems are implemented using data cube architecture. Generation of data cubes for processing user queries helps in reducing time, but has storage and data synchronization overheads. In some cases, storage overheads are so large that it becomes extremely prohibitive to generate a cube. If it is not feasible to generate a cube, the user queries are processed using on the fly aggregation. The "on the fly" aggregation demands very good query optimization, which otherwise would lead to high response time for user queries. Query optimization [31] plays a vital role in such BIS. Most of the time, the query optimization is done using techniques like indexing, but unfortunately this technique is not able to optimize aggregate queries. Hence, the idea of materialized views has been proposed to optimize such queries. A materialized view [17] is like a normal view with storage used for storing results of a view query. When a materialized view is referred, rows are directly retrieved from the storage rather than the execution of the query again, thereby reducing the processing time of the query. The stored rows of materialized view are refreshed when base tables of materialized views are updated to keep the data synchronized. Thus, materialized views have data synchronization costs which may reduce the overall advantage of improving query response time [29]. If the system has many materialized views, then the performance of the system deteriorates due to high data synchronization overheads. Therefore, it is necessary to have a minimum number of materialized views to improve the queries, which otherwise cannot be optimized by conventional methods.

The aim of this paper is to create a set of materialized views with minimum cardinality, which can optimize most of the queries. Our approach is to find frequent components in queries and create materialized views on them to optimize them. Such frequent components may represent frequent subqueries. The traditional approaches like approximate string matching algorithm [2] will not be useful here because similar queries may not have the same text. For example, if two different queries have the same subquery, then by using normal string matching technique, the queries are treated differently. Therefore, we propose a new approach of finding frequent components in queries by analyzing the query execution plan rather than query text using data mining techniques. Most of the database management systems construct query execution plan in the form of a binary tree. This paper

uses this property to build a recursive tree mining algorithm to find the frequent components of a query. In most of the applications, the query workload follows 80–20 rule, i.e., almost 20 % queries form 80 % of the work load. As the number of queries increases, the probability of a query component to repeat also increases. The creation of materialized views on these frequent components will result in the overall optimization of queries that reuse the same components. Generalized graph mining algorithms like G-Span are already developed to obtain frequent graphs from a memory dump of graphs [30]. However, this is a highly generalized algorithm and such conventional graph mining algorithms are not useful for mining the "execution plan tree components" because of the specialized nature of these trees. Node in the tree represents an operation, whereas the level of the node indicates the order in which the operation is performed. In this paper, we have proposed a new tree mining algorithm for identifying the frequent tree components in a set of such specialized trees.

The algorithm proposed in this paper analyzes multiple queries and recommends queries as well as query components. Creation of materialized views on these components will result in the optimization of all the queries having these components. Many database systems contain hundreds and even thousands of tables. Such database applications may have millions of queries [1]. These queries may have many frequent components. Mining of frequent components that can be translated as candidate queries is a challenging task [4]. The proposed tree mining algorithm for finding these components should be able to handle the work load by pruning the search space efficiently.

For a given workload, we have found that the candidate queries using our method are more in number as compared to the queries resolved using the conventional method. We have also shown that the materialized views created using candidate queries used by our algorithm show a considerable improvement in terms of "reduction of the logical block reads (GAIN MEASURE)" as a performance measure [12]. The contributions of the proposed work are as follows:

1. We have introduced the creation of a materialized view based on query components and subqueries rather than creating it only on the full query.
2. We have proposed a method of finding query components by analyzing execution plans of past query workload rather than analyzing query texts. This is a fundamental change in approach as compared to traditional methods because it helps in getting a larger set of queries which can be optimized with a lesser number of materialized views and thus improving system performance to a very large extent.
3. We have proposed new tree mining algorithms for specialized trees, which represent query execution plans to handle large query loads by providing efficient pruning techniques to reduce search space. We have also provided correctness proof of our newly designed algorithm and proved that it will mine all necessary frequent subcomponents from the given set.

4. We have done exhaustive experimentation on standard, real and synthetic workload to show that the number of candidate queries reported by our algorithm is expected to be higher than the number of candidate queries reported by the state of the art algorithms by the traditional methods.

The rest of the paper is organized as follows: Section 2 describes the related work, followed by Section 3 which describes the proposed work and the algorithm in detail. Section 4 describes how the output of the tree mining algorithm helps to create materialized views, Section 4 describes the experimentation and results and Section 5 contains the conclusion and future work.

2 RELATED WORK

View materialization is a widely-used strategy employed in data warehouses of the Business Information System to improve the performance of decision support queries. Decision support queries are highly complex in nature and make heavy use of joins and aggregations. Moreover, solving decision support queries involves computations on huge volumes of historical data, as these queries are more inclined to find trends rather than individual facts. Historical data is continuously generated by multiple fast operational OLTP (Online Transaction Processing) systems and gets accumulated in warehousing systems. Since access to a materialized view is faster than computing the views on demand, using the materialized view can speed up the analytical query processing in a data warehouse. Hence, naturally it is desirable to materialize all the possible views in a data warehouse, but this is not feasible because of resource constraints such as disk space, computation time and maintenance cost. Especially, creation of materialized views incurs an overhead after update of the base database objects which demands refreshing of all the affected materialized views.

Hence, to acquire a quick response to analytical queries within the system's resource constraints, selection of a proper set of views to materialize is an important decision while designing the warehousing system. The most commonly used technique is materializing frequent queries, which are obtained by text matching [13].

Gong in his paper [13] proposed clustering based dynamic materialized view selection algorithm (CBDMVS). It finds a cluster of SQL queries using a similarity threshold τ and if a new query's similarity is below τ for all the existing clusters, then a new cluster is formed. Similarity between queries is measured based on certain parameters like base table sets, equivalence connectivity conditions, scope connectivity conditions and output column sets. These queries are mined using text mining. CBDMVS dynamically adjusts the materialized view set, by replacing views with lowest gains where the system lacks storage space for the new query. Basically, it not only improves the overall query response time, but also reduces the computational cost that is spent while updating materialized view. Rajyalakshmi in paper [26] proposed association rule mining based materialized view selection

algorithm (ARMMVVM) for improving the performance of materialized view selection and materialized view maintenance using association rule mining. It integrates the technique of improving query response time by using frequent mining algorithm along with adjustments of the view set.

Sohrabi and Ghods in their paper [28] explored the view selection problem as a two-step process where, the first step is finding the candidate views and the second step decides the final view set from a set of candidate views under the system resource constraints such as storage space and view maintenance cost. This paper discussed the usage of Directed Acyclic Graphs (DAGs) and data cube lattice in candidate generation step, and various heuristic algorithms in the view selection step. The authors also proposed a novel algorithm based on frequent item set mining technique which aims at minimizing the view creation and maintenance cost. Paper [14] proposed a systematic review of various view selection techniques in which various techniques are compared in terms of memory storage space, cost, and query processing time. By means of this review of available literature, the authors have drawn several conclusions about the status quo of materialized view selection and a future outlook is predicted on bridging the large gaps that were found in the existing methods.

In paper [27], the authors proposed a greedy materialized view selection algorithm, which extracts query-processing and view maintenance cost related information from multiple query processing plans into a table-like structure and the algorithm also computes the optimal view set. In paper [21], the authors proposed a similarity interaction operator-based particle swarm optimization (SIPSO), in which materializing an appropriate subset of views was suggested for achieving acceptable response times for analytical queries. The proposed SIPSO-based view selection algorithm (SIPSOVSA) selects the Top-K views from a multidimensional lattice. Paper [5] proposed a game theory based framework for the materialized view selection. In the proposed framework, query processing and view maintenance costs play a game against each other as two players and continue the game until they reach the equilibrium.

The authors in paper [20] talk about a uniform query framework that can be used for traditional relational databases and NOSQL databases. This query framework can also perform joins, aggregates, filter on the data from various data sources in a single query. Hung et al. in their paper [18] proposed a cost model, having well-defined gain and loss metrics used for deciding the member views in a view set. For candidate generation, data cube is represented as lattice, and lattice is expressed in the vector form. This vector is then used to search for other dependent views. Afrati in the paper [1] addresses the problem of view selection for aggregate queries considering rewritings with multiple view sub-goals and multi-aggregate views. This paper explains how to answer aggregate queries using aggregate views by constructing equivalent rewritings and how to optimally select aggregate views to materialize, for use in those rewritings.

The authors in the paper [15] present a greedy view selection algorithm in AND/OR view graphs, which describes all possible ways to generate warehouse

views. It describes different approaches to address the view selection problem, selects the best query path which can be maximally utilized to optimize the response time of most of the queries, under the maintenance cost constraints. In the paper [25] the authors present the heuristics to determine the additional set of views to materialize under given storage constraints to reduce the overall maintenance cost of all the views. The algorithms aim at minimizing the query response time and view maintenance overheads under the given storage constraints. The paper [19] proposes a greedy-repaired genetic algorithm which selects a set of materialized cubes from the data cubes under storage space constraints, in order to reduce the amount of query cost as well as the cube maintenance cost.

In a paper [22] authors gave importance on integration of computational methods for design optimization based on data mining and knowledge discovery. This paper proposes to use radial basis function neural networks to analyze the large database generated from evolutionary algorithms and to extract the cause-effect relationship, between the objective functions and the input design variables. Gupta and Mumick in the paper [16] developed a method to deliver the optimal set of views to optimize the total query response time for a given workload under constraints that the selected set of views should incur less collective maintenance overheads than the specified amount of maintenance time. The paper proposed approximation greedy algorithm for query load having view OR graphs and A* heuristics algorithm for query load with general AND-OR view graphs. This paper has kept the storage constraints out of all the equations.

The authors in the paper [3] proposed a framework for materialized view selection that exploits data mining technique (clustering), to determine clusters of similar queries. The paper also proposed a view merging algorithm that builds a set of candidate views, by iteratively building the lattice of views. To determine the final view set, a greedy process was used where the selection criteria considered cost of storing and accessing data from views. Ashadevi in her paper [4] presented a critical survey of the past and present methodologies and solutions for the view selection problem.

Mohania and Kambayashi in the paper [24] showed that the warehouse views could be made self-maintainable if additional auxiliary relations were derived from the intermediate results of view computation in the warehouse. This paper proposed an algorithm for determining what auxiliary relations needed to be materialized to make a materialized view self-maintainable, i.e., maintainability could be viewed as an incremental process that computes the updates to both the materialized view and the additional relations.

Daneshpour and Barfourosh in their paper [9] proposed a dynamic view management system to select materialized views with new and improved architecture, which could predict incoming queries through association rule mining and three probabilistic reasoning approaches: Conditional probability, Bayes' rule, and Naïve Bayes' rule.

3 PROPOSED WORK

3.1 Problem Statement

A materialized view is a proven technique to optimize queries such as aggregate queries which otherwise cannot be optimized using conventional optimization techniques like indexing or clustering. Since materialized views have additional storage and data synchronization overheads it is better to create less number of materialized views. If a query load consists of millions of queries, it is very costly to create materialized view. It is reasonable to use less number of materialized views for optimizing computationally intensive queries.

3.2 Theoretical Analysis and Motivation

Matching query text can be used to find out frequent queries, but this approach may not work if the queries are written differently or, if several queries are doing similar operations. For example, the following queries, i.e. queries Q1 and Q2, are computationally similar to query Q3, but their textual representations are different.

Q1: select avg (salary) from emp_company group by cname;

Q2: select max (salary) from emp_company group by cname;

Q3: select max (salary), avg (salary) from emp_company group by cname.

Unlike the tree mining algorithm, the string matching algorithm will not be able to find any correlation between these queries. But, if we create materialized view for the query Q3, both the queries Q1 and Q2 will get optimized. Another example could be queries that are different, but they use the same subqueries. For example, the following queries are different, but use the same subquery.

Q4: select ename from emp_company where salary > all (select avg (salary) from emp_company);

Q5: select cname from emp_company where salary > all (select avg (salary) from emp_company).

Both the queries are using the same subquery, but they are not textually similar. Both can be optimized by creating materialized view of the subquery. The text matching algorithm which is used by traditional researchers will fail to detect such kind of commonality amongst the queries Q4 and Q5. To overcome this problem we have used the execution plan tree as a basis for finding similarity between trees and detecting similar subqueries because of the following facts.

1. If the two queries are similar, then their execution plans trees are also the same.
2. If the two queries are having the same subquery, then their execution plan trees are having the same subtree components.

With tree mining, we will be able to find a common subtree in the query plan trees of Q4 and Q5. Therefore, the problem statement is defined as "To find a set of materialized view queries" [4] from the existing query load by mining database of query execution plans such that:

1. The set of materialized views should optimize a large number of time consuming queries.
2. The set should have low cardinality to avoid storage overheads and data synchronization costs.

Once "execution plan tree" is decided as the basis of similarity then the next challenge is to design algorithm which efficiently mines frequent tree components from the set of millions of trees. General frequent itemset mining or graph mining algorithms cannot be applied because of specialized nature of plan trees. So the specialized frequent tree algorithm is designed by providing heuristics and correctness proof of the algorithm is provided.

3.3 Terminologies and Examples

In this section, we first define a few terminologies used for explaining our tree mining algorithm which is used to mine database of query execution plans which are in the form of binary trees.

View: A view is a derived relation, defined by a query in terms of base relations and/or other views.

Materialized view: A view is said to be materialized if its query result is persistently stored, otherwise it is said to be virtual. We refer to a set of selected views to materialize as a set of materialized views.

Workload: A workload or a query workload is a given set of queries, $Q = \{Q_1, Q_2, \ldots, Q_n\}$. Each query in the query workload can be described using its frequency and cost of execution.

View selection: Given a database schema and a query workload, the objective is to select an appropriate set of materialized views to improve performance of database in processing the workload, i.e. in executing queries in the workload. The ideal view set can comprise queries which are useful in optimizing the performance of a large number of queries in the workload.

Tree: Tree is a directed acyclic graph denoted as $T(R, V, L, E)$, where V is the set of nodes in T; R is one of the node of T and is the root of T; L is the set of the labels of the nodes; and E is the set of directed edges in T. All trees considered in this paper are rooted labeled trees.

Query and query plans: Every query has a query plan associated with it. A query plan shows the execution path of the query [12]. Most of the Database Management Systems (DBMS) use binary trees to represent the query execution plans

where leaves indicate the data source and node indicates the type of operation such as join. Though we have designed an algorithm based on the assumption that the query plan is in the form of binary tree, the algorithm can be easily extended for generalized plan tree. Figure 1 indicates query plan for the following query.

Select e.ename, e.city from employee e where e.ename in (select c.ename from emp_company c where c.cname = 'ACC' and c.salary > (select avg (salary) from emp_company)).

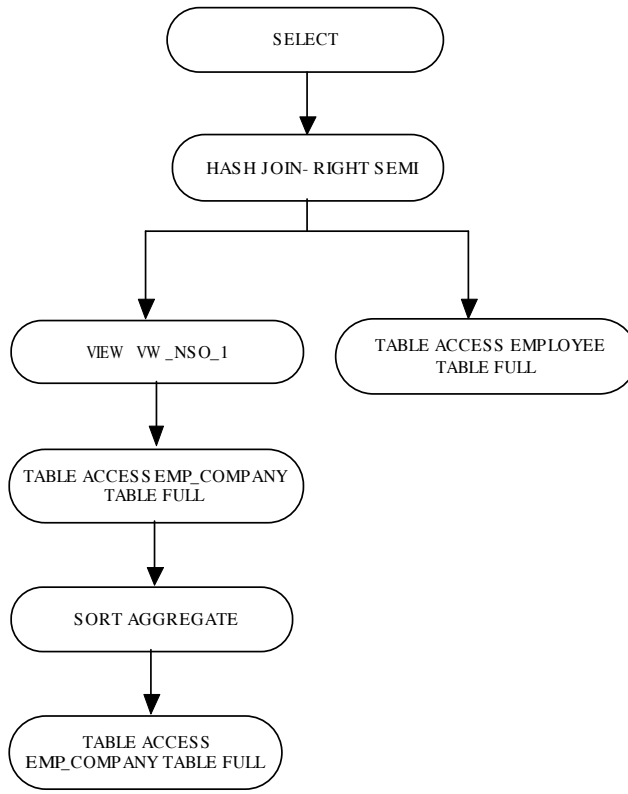


Figure 1. Query plan for the query

Each node in the query plan tree represents an operation. We extract these query plans from all the queries present in the query load. These plans constitute the tree database (TDB) which is used as an input for the tree mining algorithm.

3.3.1 Query Subtree and Supertree

Consider the two trees $T(R, V, L, E)$ and $T'(R', V', L', E')$ based on tree definition in Section 3.2.5. Assuming T' as a subtree (embedded tree) of T ($T' \subset T$). It implies that $V' \subset V$, $E' \subset E$, $L' \subset L$, $L'(V) = L(V)$. If $(v_1, v_2) \in E'$ and v_1 is the ancestor of v_2 in T, then it is preserved in T' also. If T' is subtree of T then T is called a supertree of T'.

In our paper we are considering only those embedded trees whose set of leaf nodes is a subset of leaf nodes of a tree. Any embedded tree which does not terminate strictly at the leaf level of an enclosing supertree is not considered a valid subtree. Only such subtrees represent a valid query component like subquery or part of query on which materialized view can be created. Figure 2 indicates the subtree.

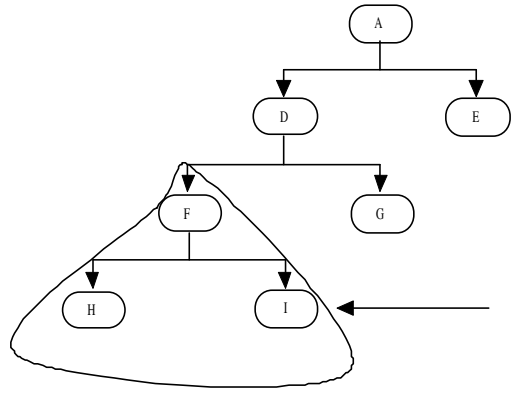


Figure 2. Subtree which represents a subexpression or a subquery

3.3.2 Tree Database (TDB)

It is a set of query execution plan trees collected from traces of database management system. The dataset containing query plans is in the form of binary trees. This data can be easily obtained from the trace utility provided by database management systems. The trace utilities normally dump data in relational tables which includes sql query text, query execution plans. Each query plan in the TDB is associated with an identifier (SQL_ID). It also provides information like number of logical reads, cost of query and number of executions of each query. All such information is available in the dynamic dictionary views which can be easily extracted. The tree database which is extracted from such views is referred to as the query workload.

3.3.3 Support

Given the tree database TDB, assuming a tree $T \in TDB$, and S' is a subtree of T, then support of subtree S' equals to the number of instances of S' in TDB including

the instance(s) in T. The task of frequent subtree mining from TDB with given minimum support σ is to find all the candidate subtrees that has support at least equal to σ. The support for subtree given in Figure 3 in the database given in Figure 5 is 2.

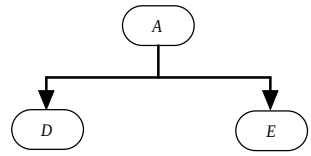


Figure 3. Support of a subtree (i.e. a subquery) is the number of its occurrences in all the query plans

The subtree A-D-E is present in two trees shown in Figures 5 and 6.

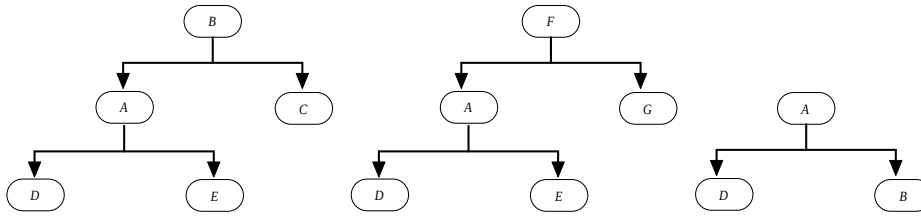


Figure 4. Sample tree dataset (4.1, 4.2, 4.3)

3.3.4 Threshold and Frequent Subtree

It is the minimum support value which qualifies the subtree to be get included into a list of frequent sub-trees. If the support of a sub-tree is greater than or equal to the specified threshold (% value of total trees), then that subtree is called a frequent subtree. The support of subtree S given in Figure 3 is 2. If threshold is 2, then sub-tree S is frequent. If the minimum support value is decreased then more trees will be qualified as frequent trees and more number of materialized views will be created. If more materialized views are present then the high synchronization cost of the materialized views will have adverse impact on performance so the minimum support value should be carefully chosen according to requirement of application. In this paper, for experimentation, the support value is chosen as 50 % so that the optimization is done with less number of materialized views. If the application is having a large number of transactions then this value can be increased to create less number of views, and if the application is having only a large number of retrievals where materialized view synchronization cost is negligible then the support count can be decreased to optimize more queries using materialized views. The value of 50 % allows the user to check performance and adjust it according to the nature of the application.

3.3.5 Maximal Frequent Subtree

The subtree S of any tree T is said to be maximal frequent subtree if S is frequent and there is no supertree S' of S in tree T such that S' is frequent. In other words, there does not exist any frequent tree whose subtree is S. If materialized view is created on maximal frequent subtree then large part of the query is optimized [30].

3.3.6 GAIN Measure (GM)

It is percentage reduction in the number of logical block reads after a query set is optimized using materialized views. For example, a user fires a query "Select sum (salary) from employee group by department_name" then it will require 2 000 logical block reads if the table "employee" occupies 2 000 blocks on the disk. If the materialized view is created on the same query then it will store department wise salary which will take very less space on the disk. If the size of materialized view is 10 blocks then the percentage reduction (GM) is $(1\,990/2\,000) * 100$.

3.4 Tree Mining Algorithm

The proposed algorithm for mining frequent subtrees uses a recursive bottom up approach, i.e., it traverses the search space in a bottom up manner starting from the leaf. The method to mine the components is given below.

Input
TDB: Set of query execution plans
δ: Support Threshold
Output
FTDB: The list of frequent components (Subtrees) in trees and list of queries associated with them.

3.4.1 Preprocessing

As, all the queries do not require optimization using materialized view, queries which have very less cost, or which are very infrequent do not need to be optimized using materialized views. To reduce the query load, the following preprocessing is done.

- Query cost is calculated in terms of the number of logical block reads. The query is placed in the experimental load if (number of logical reads) $*$ (frequency of query) is greater than some threshold (theta).
- If the base tables of queries are frequently updated then refresh cost of materialized view is high and such queries can be removed from experimental load.
- Within experimental load the queries are ordered using level of execution plan tree.

Variables
TreeNode: It is a data structure pointing to the node of the tree. All nodes in a tree that are the children of TreeNode are accessible from this data structure. In fact, each tree is represented by its root TreeNode.
TreeNode → Left: Left node of TreeNode
TreeNode → Right: Right node of TreeNode
N: number of query plans in dataset TDB
Tr: Parameter required for Threshold Pruning. $Tr = N * ((1 - \sigma) \div 100)$
FrequentTreeMap<FrequentTree, List_Of_Sql_Ids>: Map of frequent subtrees, present along with their sql ids in which the subtrees are present.
TreeList: List in which all query plans are stored in the form of binary trees. In this list, all elements are TreeNodes [refer]. All TreeNodes are pointing to root of trees.
SQL_Id_List (T): List of all SQL_IDs of which sql queries that includes component subtree T.
MaximalFrequentSubtreeList: It is list containing all maximal frequent subtrees encountered.

Algorithm: FrequentTreeMiner
Preprocessing;
FrequentTreeMap = ();
MaximalFrequentSubtreeList = ();
begin
for $i = 0$; $i < N$; i++ **do**
    MineSubTree(TreeList[i]); ▷ (1)
end for

3.4.2 Analysis

For each query execution plan, all components are searched and if the component is frequent then it is inserted in the frequent tree set as given in step (4) of the algorithm. For each component, searching is done by calling function searchsubtree () in step (6), which tries to find out whether the tree component is a subtree of the existing tree and then the database of all the search trees is scanned as given in

Function: MineSubTree ▷ The function MineSubTree returns a query list which contains the subtree T only if T is frequent subtree, otherwise it returns a null list.
Input
TreeNode: Tree which is to be tested as maximal subtree
Output
FrequentTreeMap: List of frequent subtrees
Status: Boolean variable to indicate whether specified tree is maximal subtree.

```
Function MineSubTree (TreeNode) return boolean
if T is null then
    return true
else
    if T is already subtree of any tree in MaximalFrequentSubtreeList then ▷ (2)
        return true
    else
        if MineSubTree (T → left) and MineSubTree (T → right) then ▷ (3)
            if CheckSubtreeIfFrequent (T) then
                FrequentTreeMap. Insert (T, SQL_Id_List(T)); ▷ (4)
                return true
            else
                return false
            end if
        end if
    end if
end if
```

Function: CheckSubtreeIfFrequent (TreeNode) ▷ The function CheckSubtreeIfFrequent (TreeNode T) will check if support of the T is greater than threshold, i.e., if T is frequent.
Input
TreeNode: Tree representing query plan or component of query plan
Output
Status: Boolean variable indicating whether the given tree is frequent.

```
Function CheckSubtreeIfFrequent (TreeNode T) return boolean
startIndex = index for tree next to tree that contains component T.
count = 1;
for i = startIndex; i < N; i++ do ▷ (5)
    if searchSubtree (TreeList[i], T) == True then ▷ (6)
        count = count + 1 ▷ (7)
    end if
end for
if count >= ceil (Tr * N) then return true; else return false;
end if
```

Function: searchSubtree (TreeNode T, TreeNode subT)
This function checks whether any tree subT is present in tree T (subT represents the TreeNode, i.e., root node of tree to be searched as explained earlier and similarly, T represents TreeNode, i.e., root of tree in which subT is to be searched).

step (5). Step (1) executes N (size of query database) times while step (3) which recursively searches for each component in tree executes $(L-1)$ times where L is level of the tree. For each component, the database of query plans is searched in step (5) which will also be executed N times and each such search at step (6) takes time proportional to $(L-1)$. Therefore, the total complexity of the algorithm is $(N*(L-1))^2$. To get all frequent components, the usual method is to enumerate all components that are in the dataset, and count the support of these item sets, and decide whether they are frequent or not. However, when the number of distinct items is huge, the algorithm that explores the entire search space may be inefficient due to the exponential increase in permutations. To avoid this problem, we employ a few techniques to reduce the search space. Here L, level of the tree, is dependent on the user query and it cannot be controlled, therefore the only way to reduce the cost of the algorithm is to reduce "N" by considering only queries which require performance improvement. Thus, N can be reduced using the following steps.

(1) Threshold pruning: While determining whether a new component is frequent or not, after verifying T_r trees [where $T_r = N*(1-\text{Threshold}/100)$] without a single instance of a new component, the new component is automatically considered infrequent. A component which does not occur even once in T_r trees (where T_r is $T_r = N*(1-\text{Threshold}/100)$ and N is the number of trees in the dataset) is considered infrequent. This is because even if it occurs in every tree after T_r trees, it still will not cross the threshold. Thus, new components are not mined after T_r trees. This step can be introduced after step (7) as follows. If $(count < T_r)$ break.

(2) Bottom up pruning: If the component containing two children of a tree node are infrequent then the component containing their parent will also be infrequent. So, while parsing any tree, suppose we find one node infrequent, we do not need to consider its parent, resulting in pruning the search space significantly. This is implemented in step (3).

(3) Maximal frequent lookup pruning: If a component is already frequent then there is no need to check it again as a lookup list of all maximal frequent subtrees is maintained. Every time a component is encountered, it is first searched in the look up list to check if it is a subtree of any other maximal frequent subtree, if it is found in the list, no further checks are carried out to find it is frequent or not, thereby reducing the search space drastically. In absence of the above mentioned look up lists, for every instance of each frequent component, we would have to search in the complete tree database resulting in huge inefficiency for large datasets. This is implemented in step (2).

(4) Filtering based on data source: Step (3) ensures that all the query components are searched and "for loop" specified in step (1) ensures that the whole database of query plans is searched so that the algorithm ensures that if some component is frequent then it is stored in the frequent set. If the component is not frequent then it will not be inserted in the frequent set as given in step (4).

The algorithm ensures that the output frequent set contains all possible frequent components in the database and components that are not frequent will never get a place in the output set.

3.5 Candidate Queries for Materialized View Creation

The aim of the proposed method is to find candidate queries for generating the materialized views, which otherwise cannot be obtained by conventional state of the art methods. The tree mining algorithm discussed in the previous section gives frequently occurring subtrees (components) and list of queries associated with each frequent subtree as an output. These frequent components are then analyzed and used to create materialized views. The following examples shows how the candidate queries are obtained by the algorithm which otherwise cannot be obtained by conventional methods. For example, suppose there are two queries:

Query 1: select e.cname from emp_company e where e.salary > (select avg (salary) from emp_company) group by cname;

Query 2: select e.ename, e.city from employee e where e.ename in (select c.ename from emp_company c where c.cname = 'ACC' and c.salary > (select avg (salary) from emp_company)).

The execution plans of both the queries are shown in Figures 5 and 6, respectively, with the frequent component associated with them.

The materialized view mv2 is created on this frequent component as of the queries on the schema of the data warehouse which contains five dimension tables PRODUCTS:

```
select cname, max(salary), avg(salary), sum(salary), min(salary),
count(salary) from emp_company group by cname.
```

The materialized view mv2 will optimize both queries.

4 EXPERIMENTAL EVALUATION AND RESULTS

The experimentation was performed on a 2.3 GHz Intel core i5 processor with 4 GB main memory, running on Mac OS X. This experimentation was done using standard query workload mentioned in [26] on Oracle 11g Database Management System. The workload consisted of TIMES, CHANNELS, PROMOTIONS, CUSTOMERS having 15, 31, 4, 8 and 15 attributes (columns), respectively. It also contains one FACT table named SALES which contains two measures "QUANTITIES_SOLD" and "AMOUNT_SOLD". The performance on standard workload is compared to the recent established algorithms MVFI [26], ARMMVVM [13] and CBMVS [28] using GAIN measure (GM) [28] as a performance criterion. The optimization tests were carried out using standard query workload of data warehouse in which size is

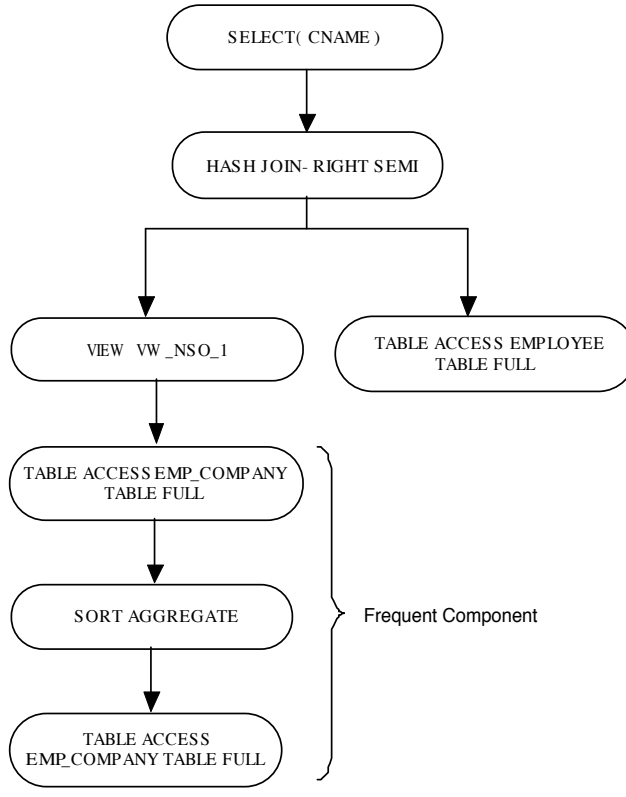


Figure 5. Query plan for Query1

Sr. No.	Dataset Size	View Selection Algorithms (Gain Measure)			
		MVFI	ARMMVVM	CBMVS	Proposed
1	0.5 GB	5.5	5	4.4	6.1
2	1.0 GB	11	9	7.7	12.5
3	1.5 GB	17.4	13	10.5	19.8
4	2.0 GB	22.4	18	14.5	25.1

- MVFI – Materialized view selection based on frequent itemset mining algorithm.
- ARMMVVM – An association rule mining for materialized view selection.
- CBMVS or CBDMVS – Clustering based dynamic materialization view selection algorithm.
- GM – Gain Measure.

Table 1. Comparison between recent algorithms with proposed algorithms on the standard query workload

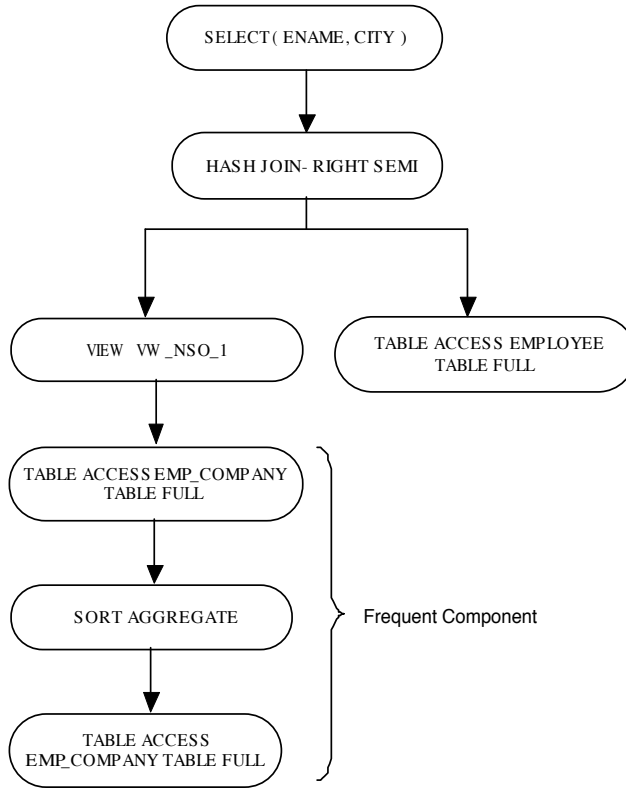


Figure 6. Query plan for Query2

varied from 0.5 GB to 2 GB by controlling the number of rows in table SALES. The result of the experimentation is given in Table 1.

The experimentation results in Table 1 indicate that there is large improvement in GM compared to recent methods for all sizes of query work load by the proposed method. We have also used synthetic and real life datasets for the experimentation, to test the applicability of proposed method on varying types of queries. The real data set is obtained from Management Information System of National Institute of Technology, Nagpur. The majority of queries in almost all real life applications consist of time consuming operations such as joins, aggregations and groupings, hence we have taken query load which is a mixture of queries having such operations. We have composed three query sets QS1, QS2 and QS3 having different composition of join and aggregate queries. The tables referred in the queries use different columns which are aggregated and grouped. We make sure to have more variations in datasets in the form of aggregations and joins. The composition of data sets is described in Table 2. The first row indicates query set QS1 executed on real database of "MIS-

VNIT" which contains 2087 queries having 48 % join queries, 17 % aggregate queries and 29 % queries using both joins and aggregates. The other query sets are shown in successive rows.

Sr. No.	QL	DS	N	QJ	QA	QJA
1	QS1	Real MIS-VNIT	2 087	48	17	29
2	QS2	Synthetic	3 086	26	28	35
3	QS3	Synthetic	2 809	20	39	32

- QL – Query Load
- DS – Data Source
- N – Number of Queries (Total number of complex queries in the query workload)
- QJ – Percentage of queries involving only joins
- QA – Percentage of queries involving only aggregations
- QJA – Percentage of queries involving both joins and aggregations

Table 2. Dataset characteristics

The datasets were cached in the main memory during the algorithms processing stage, to avoid high data access costs. The numbers of frequent trees which are to be mined are controlled by parameter "frequency threshold". If the threshold is higher, then the algorithm produces less frequent trees and lesser number of materialized views are created. Since the number of materialized views cannot be large because of synchronization overheads, we have done the experimentation by setting the threshold to 50 % of the total candidate trees (threshold is taken as 50 % with the assumption that around 50 % of the total workload will be having frequent patterns. If more queries are to be optimized then the threshold can be reduced). The performance is measured using GM. The performance results are shown in Table 3 with different query loads. The proposed tree mining algorithm is implemented in Java.

QL	LR	View Selection Algorithms (Gain Measure)			
		MVFI	ARMMVVM	CBMVS	Proposed
QS1	3 560 642	24.34	21.21	15.38	40.62
QS2	4 701 867	33.68	29.24	27.45	37.80
QS3	3 857 673	31.25	28.41	24.37	40.10

- QL – Query Load
- LR – Logical Reads before creation of materialized views

Table 3. Results showing comparative analysis of best known algorithms with proposed algorithm on real and synthetic datasets described in Table 2

From Table 3, it is interpreted that the GM is considerably increased with the proposed tree mining algorithm when compared to state of the art algorithms in all types of query load because the proposed algorithm is designed to mine frequent queries as well as frequent subquery components. It has also been observed that

for the large datasets of query workload of the size 4 million, the improvement is considerable.

4.1 Scalability

The scalability of the tree mining algorithm was analyzed by using query load of three different sized datasets. The query load is obtained by using queries mentioned in Table 2 multiple times. It was found that due to efficient pruning techniques, the processing time increased linearly with size, though the worst case complexity could be $O(N^2)$. The reason behind the experimental linearity is that if the tree is already a part of the frequent tree, then the cost of finding the frequency of the tree or the database scan is minimized.

Another reason for linearity could be that if a subtree is not frequent then its supertree is also not frequent, hence there is no cost of extra database scan. The performance of the algorithm is given in Table 4. In general, execution plans for groupings and aggregations have trees of larger length and hence mining frequent components takes more time. Since the algorithm is executed offline without any hard time constraint, practically the execution time mentioned in the table is well within acceptance level.

Datasets	No. of Queries in the Dataset			
	100 000	200 000	300 000	400 000
QS1	381	708	1 020	1 343
QS2	335	592	829	1 087
QS3	467	931	1 330	1 683

Table 4. Execution time of tree mining algorithm on different datasets with different sizes (number of queries) of query load (time in seconds)

5 CONCLUSION AND FUTURE WORK

It is a challenging task to select a set of queries from a huge query load, for creating materialized views. This is because such a set should not only be small, but should also provide maximum benefit for optimizing most of the queries. Most of the earlier methods rely on approximate text matching algorithms or finding frequent patterns in queries which refers to same set of tables. Such an approach may not work if frequent queries appear as sub queries.

In this paper, an attempt has been made to find frequent queries as well as frequent subqueries. The proposed method uses "query execution plan" for finding frequent query components instead of operating on query text. Such query plan is represented as a binary tree which can then be extracted from dynamic dictionary views which are provided by most of the databases and data warehouse systems, making the proposed method feasible. In the proposed method, finding frequent

components in a large set of queries is translated as finding frequent subtrees, and efficient algorithms are proposed to extract subtrees with the correctness proof of the algorithm. The proposed method suggests various pruning techniques to effectively reduce the search space and to combat the huge query load. Certain queries which do not require materialized views are preprocessed and removed from the experimental load. The proposed method is compared with standard workload mentioned in the literature and its performance is compared with the recent methods available in the literature. The experimental evaluation indicates that the proposed method gives better performance than all the recent methods irrespective of query load size. The detailed study is done on real and synthetics data sets to check the performance on various types of workloads. The experimental evaluation indicates that the performance is improved to very large extent by the proposed method in all types of query workloads.

In the future, the selected queries can be analyzed using data synchronization costs of materialized views and total optimization can be done considering reduction of query cost and increase in data synchronization costs. The queries having high data synchronization costs can be modified and optimized using conventional methods and can be pruned in preprocessing steps.

REFERENCES

[1] Afrati, F. N.: Determinacy and Query Rewriting for Conjunctive Queries and Views. Theoretical Computer Science, Vol. 412, 2011, No. 11, pp. 1005–1021, doi: 10.1016/j.tcs.2010.12.031.

[2] Al-Khamaiseh, K.—ALShagarin, S.: A Survey of String Matching Algorithms. International Journal of Engineering Research and Applications, Vol. 4, 2014, No. 7, pp. 144–156.

[3] Aouiche, K.—Jouve, P. E.—Darmont, J.: Clustering-Based Materialized View Selection in Data Warehouses. In: Manolopoulos, Y., Pokorný, J., Sellis, T. K. (Eds.): Advances in Databases and Information Systems (ADBIS 2006). Springer, Berlin, Heidelberg, Lecture Notes in Computer Science, Vol. 4152, 2006, pp. 81–95, doi: 10.1007/11827252_9.

[4] Ashadevi, B.: Analysis of View Selection Problem in Data Warehousing Environment. International Journal of Engineering and Technology, Vol. 3, 2012, No. 6, pp. 447–457.

[5] Azgomi, H.—Sohrabi, M. K.: A Game Theory Based Framework for Materialized View Selection in Data Warehouses. Engineering Applications of Artificial Intelligence, Vol. 71, 2018, pp. 125–137, doi: 10.1016/j.engappai.2018.02.018.

[6] Chaudhuri, S.—Krishnamurthy, R.—Potamianos, S.—Shim, K.: Optimizing Queries with Materialized Views. Proceedings of the Eleventh IEEE International Conference on Data Engineering (ICDE '95), Taiwan, 1995, pp. 190–200, doi: 10.1109/ICDE.1995.380392.

[7] Chen, D.—Chirkova, R.—Sadri, F.: Query Optimization Using Restructured Views: Theory and Experiments. Information Systems, Vol. 34, 2009, No. 3, pp. 353–370, doi: 10.1016/j.is.2008.10.002.

[8] Chi, Y.—Xia, Y.—Yang, Y.—Muntz, R. R.: Mining Closed and Maximal Frequent Subtrees from Databases of Labeled Rooted Trees. IEEE Transactions on Knowledge and Data Engineering, Vol. 17, 2005, No. 2, pp. 190–202, doi: 10.1109/TKDE.2005.30.

[9] Daneshpour, N.—Barfourosh, A. A.: Dynamic View Management System for Query Prediction to View Materialization. International Journal of Data Warehousing and Mining, Vol. 7, 2011, No. 2, pp. 67–96, doi: 10.4018/jdwm.2011040104.

[10] Deshpande, P. S.: Data Warehousing Using Oracle. Wiley-DreamTech Press, India, 2005.

[11] Afrati, F.—Chirkova, R.: Selecting and Using Views to Compute Aggregate Queries. Journal of Computer and System Sciences, Vol. 77, 2011, No. 6, pp. 1079–1107, doi: 10.1016/j.jcss.2010.10.003.

[12] Goldstein, J.—Larson, P.-Å.: Optimizing Queries Using Materialized Views: A Practical, Scalable Solution. ACM SIGMOD International Conference on Management of Data, Santa Barbara, California, USA. ACM SIGMOD Record, Vol. 30, 2001, No. 2, pp. 331–342, doi: 10.1145/376284.375706.

[13] Gong, A.—Zhao, W.: Clustering-Based Dynamic Materialized View Selection Algorithm. Proceedings of the 2008 Fifth International Conference on Fuzzy Systems and Knowledge Discovery (FSKD 2008), Vol. 5, 2008, pp. 391–395, doi: 10.1109/FSKD.2008.96.

[14] Gosain, A.—Sachdeva, K.: A Systematic Review on Materialized View Selection. In: Satapathy, S., Bhateja, V., Udgata, S., Pattnaik, P. (Eds.): Proceedings of the 5th International Conference on Frontiers in Intelligent Computing: Theory and Applications. Springer, Singapore, Advances in Intelligent Systems and Computing, Vol. 515, 2017, pp. 663–671, doi: 10.1007/978-981-10-3153-3_66.

[15] Gupta, A.—Mumick, I. S.: Maintenance of Materialized Views: Problems, Techniques, and Applications. IEEE Data Engineering Bulletin, Vol. 18, 1995, No. 2, pp. 3–18.

[16] Gupta, H.—Mumick, I. S.: Selection of Views to Materialize Under a Maintenance Cost Constraint. In: Beeri, C., Buneman, P. (Eds.): Database Theory – ICDT '99. Springer, Berlin, Heidelberg, Lecture Notes in Computer Science, Vol. 1540, 1999, pp. 453–470, doi: 10.1007/3-540-49257-7_28.

[17] Gupta, H.—Mumick, I. S.: Selection of Views to Materialize in a Data Warehouse. IEEE Transactions on Knowledge and Data Engineering, Vol. 17, 2005, No. 1, pp. 24–43, doi: 10.1109/TKDE.2005.16.

[18] Hung, M.-C.—Huang, M.-L.—Yang, D.-L.—Hsueh, N.-L.: Efficient Approaches for Materialized Views Selection in a Data Warehouse. Information Sciences, Vol. 177, 2007, No. 6, pp. 1333–1348, doi: 10.1016/j.ins.2006.09.007.

[19] Hylock, R.—Currim, F.: A Maintenance Centric Approach to the View Selection Problem. Information Systems, Vol. 38, 2013, No. 7, pp. 971–987, doi: 10.1016/j.is.2013.03.005.

[20] Karanjekar, J. B.—Chandak, M. B.: Uniform Query Framework for Relational and NoSQL Databases. Computer Modeling in Engineering and Sciences, Vol. 113, 2017, No. 2, pp. 177-187, doi: 10.3970/cmes.2017.113.177.

[21] Kumar, A.—Vijay Kumar, T. V.: Improved Quality View Selection for Analytical Query Performance Enhancement Using Particle Swarm Optimization. International Journal of Reliability, Quality and Safety Engineering, Vol. 24, 2017, No. 6, Art. No. 1740001, doi: 10.1142/S0218539317400010.

[22] Lian, Y. S.—Liou, M. S.: Mining of Data From Evolutionary Algorithms for Improving Design Optimization. CMES: Computer Modeling in Engineering and Sciences, Vol. 8, 2005, No. 1, pp. 61–72, doi: 10.3970/cmes.2005.008.061.

[23] Lovis, C.—Baud, R. H.: Fast Exact String Pattern-Matching Algorithms Adapted to the Characteristics of the Medical Language. Journal of the American Medical Informatics Association, Vol. 7, 2000, No. 4, pp. 378–391.

[24] Mohania, M.—Kambayashi, Y.: Making Aggregate Views Self-Maintainable. Data and Knowledge Engineering, Vol. 32, 2000, No. 1, pp. 87–109, doi: 10.1016/S0169-023X(99)00016-6.

[25] Ross, K. A.—Srivastava, D.—Sudarshan, S.: Materialized View Maintenance and Integrity Constraint Checking: Trading Space for Time. Proceedings of the ACM SIGMOD International Conference on Management of Data (SIGMOD '96), Montreal, Quebec, Canada. ACM SIGMOD Record, Vol. 25, 1996, No. 2, pp. 447–458, doi: 10.1145/235968.233361.

[26] Vishwanath, R. P.—Rajyalakshmi, R.—Reddy, S.: An Association Rule Mining for Materialized View Selection and View Maintenance. International Journal of Computer Applications, Vol. 109, 2015, No. 5, pp. 15–20.

[27] Sohrabi, M. K.—Azgomi, H.: TSGV: A Table-Like Structure-Based Greedy Method for Materialized View Selection in Data Warehouses. Turkish Journal of Electrical Engineering and Computer Sciences, Vol. 25, 2017, No. 4, pp. 3175–3187, doi: 10.3906/elk-1608-112.

[28] Sohrabi, M. K.—Ghods, V.: Materialized View Selection for a Data Warehouse Using Frequent Itemset Mining. Journal of Computers, Vol. 11, 2016, No. 2, pp. 140–148, doi: 10.17706/jcp.11.2.140-148.

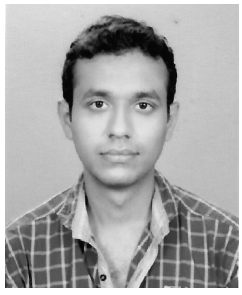
[29] Yang, J.—Karlapalem, K.—Li, Q.: A Framework for Designing Materialized Views in Data Warehousing Environment. Proceedings of the 17th International Conference on Distributed Computing Systems, IEEE, Baltimore, MD, USA, 1997, pp. 458–465, doi: 10.1109/ICDCS.1997.603380.

[30] Yan, X.—Han, J.: gSpan: Graph-Based Substructure Pattern Mining. Proceedings of the 2002 IEEE International Conference on Data Mining, Maebashi City, Japan, IEEE, 2002, pp. 721–724, doi: 10.1109/ICDM.2002.1184038.

[31] Yousri, N. A. R.—Ahmed, K. M.—El-Makky, N. M.: Algorithms for Selecting Materialized Views in a Data Warehouse. Proceedings of the 3rd ACS/IEEE International Conference on Computer Systems and Applications, IEEE, Cairo, Egypt, 2005, doi: 10.1109/AICCSA.2005.1387024.

Atul Thakare received his post-graduate degree (M.Eng. (CSE)) from SGB Amravati University, Maharashtra, India, and his undergraduate degree (Bc.Eng. (CT)) from RTM Nagpur University, Maharashtra, India. Currently, he is pursuing his Ph.D. from VNIT, Nagpur, Maharashtra, India. He has a total of 16 years experience, six years in the IT industry and ten years in academic profession.

Srijay Deshpande has received his B-Tech from Visvesvaraya National Institute of Technology, Nagpur, M-Tech from IIT-Bombay, and currently he is working as Data Scientist at Microsoft Private Ltd., India.



Amit Kshirsagar has received his B-Tech from Visvesvaraya National Institute of Technology, Nagpur, and Masters from the University of Florida and currently he is working as Senior Software Developer at Visa Inc., (USA) in Cyber Security.

Parag Deshpande has received his Ph.D. from Nagpur University, Nagpur, India and his M-Tech from IIT Powai, Mumbai, India. He is currently working as Professor in the Department of Computer Science and Engineering, VNIT, Nagpur, Maharashtra, India. He has 31 years of academic experience. He is the author of several books including C and Data Structure, Data Warehousing Using Oracle, SQL and PL/SQL for Oracle 11g. He is a member of ISTE and SAE-India.

Computing and Informatics, Vol. 38, 2019, 497–524, doi: 10.31577/cai_2019_2_497

MODEL VARIATIONS AND AUTOMATED REFINEMENT OF DOMAIN-SPECIFIC MODELING LANGUAGES FOR ROBOT-MOTION CONTROL

Verislav Djukić

Djukic Software GmbH
Nürnberg, Germany
e-mail: info@djukic-soft.com

Aleksandar Popović

Faculty of Science, University of Montenegro
Podgorica, Montenegro
e-mail: aleksandarp@rc.pmf.ac.me

Ivan Luković, Vladimir Ivančević

Faculty of Technical Sciences, University of Novi Sad
Novi Sad, Serbia
e-mail: {ivan, dragoman}@uns.ac.rs

Abstract. This paper presents an approach to handling frequent variations of modeling languages and models. The approach is based on Domain-Specific Modeling and linking of modeling tools with adaptive Run-Time Systems. The applicability of our solution is illustrated on an example of domain-specific languages for robot control. Special attention was given to the following problems: 1) model-level debugging; 2) performing fast transformation of models to native code for various hardware platforms and operating systems; and 3) specification of views and view-based generation of applications for validation of meta-models, models, and generated code. The feedback for automated refinement of models and meta-models is provided by a custom adaptive Run-Time System. For the purpose of synchronizing models, meta-models, and the target Run-Time System, we introduce action reports, which allow model-level debugging. In order to simplify handling of frequent model variations, we have introduced the linguistic concept of a modifier.

Keywords: Domain-specific modeling, run-time system, model variations, model execution, model-level debugging

Mathematics Subject Classification 2010: 68Q60, 68T40

1 INTRODUCTION

In this paper, we communicate our experience concerning the development and application of Domain-Specific Modeling (DSM) and adaptive Run-Time Systems (RTS) for robot control. We present typical problems and our solutions related to the practice of:

1. constructing and applying robot-motion control languages (RMCL);
2. frequent variations of robot control models;
3. model execution and model-level debugging, i.e., incremental updating of control logic; and
4. parallel refinement of robot control models and modeling languages.

Our approach, which is based on the DSM architecture [1], is named the DVMEx Approach. In order to verify the approach in practice, we developed run-time systems, compilers, interpreters, and modeling tools, all of which comprise the DVMEx IDE. Besides using our components to verify the approach, we also employed MetaEdit+ WB and MetaEdit Modeler [2]. These tools have been successfully used in various domains, e.g., automation, control and embedded systems.

There is a need for significant improvements of software development in automation and robot control, especially in the development of tools for:

1. formal specification and execution of control processes, and
2. construction and application of RMCLs.

For each level in the architecture of DSM solution, we specify one of the most important problems.

The level of the modeling languages. General purpose graphical languages (e.g., UML) are often used for modeling of specific processes, but they are not sufficiently understandable to users in a particular application domain.

The level of the model transformations. Transformations are complicated for an average programmer. Moreover, they are focused, or even limited, to a single target general purpose programming language (GPL).

The level of the target interpreter or run-time system. The target interpreter in most cases does not contain meta-data describing the semantics of a control process, but only commands concerning the semantics of basic logical and arithmetic operations. A single invalid basic operation may lead to

an unexpected or unresolvable state. Parsers used for reverse construction of class diagrams or state diagrams are part of most UML tools, but they do not solve the problem of losing the relationships between the modeling tools and the target interpreter of specification.

The DVMEx Approach solves these problems in the following manner:

The level of the modeling languages. Instead of GPLs and existing robot-motion control languages or operating systems, such as Robot Operating System (ROS [3]), more DSLs are constructed and used. Over the DSL models, three views are initially defined, one of which is focused on the topological properties of a robot arm, the other on motion and control logic, and the third on the real-world environment where the robot performs actions. Due to the usage of DSLs, all these views are close and comprehensible to end-users, domain specialists, and software architects. Figure 1 shows the three views on the robot control model. Three different DSLs, which are integrated at the meta-model level, are used simultaneously for the modeling robot task. The first language, which is presented in the left-hand side in Figure 1, is aimed at specification of topological properties of a robot arm, such as number of joints and fingers, length of each segment, constraints for rotation and elevation, etc. The second language, which is presented in the middle of Figure 1, is used for description of a state machine, i.e., an initial set of actions and states, as well as their relations with signals coming through various sensors, and commands explicitly issued by end-users. Since the function-block language (IEC 61131-3) is used in the automation and industry, it is convenient that this DSL uses function-block diagrams with graphical syntax that is close to users from the concrete application domain. The third language and submodel (right-hand side in Figure 1) are used for description of environment in which a robot performs actions. It is to the greatest extent specific for the application domain. When it comes to the drawing of portraits or sketches, this language is used to specify motions, canvas dimensions, canvas distance, rotation and elevation in 3D space, relative to the reference point of the robot arm. When there are obstacles between the canvas and robot arm, then the DSL for motion specification should include concepts for expressing the effects of obstacles on motion. Figure 1 presents an example of a motion variation, where curve is shifted upward, and then rotated.

The first and second DSLs are being constructed quickly in practice, and the validity of the specification can be more easily verified than in the third language. A simple construction of language concepts for 3D representation of motion is not expected from a general purpose DSM tool. Therefore, this specific problem is solved by using action reports [4] and more advanced libraries or 3D visualization components, which are not part of DSM tools. Additional descriptions may be found in the section devoted to the model-level debugging.

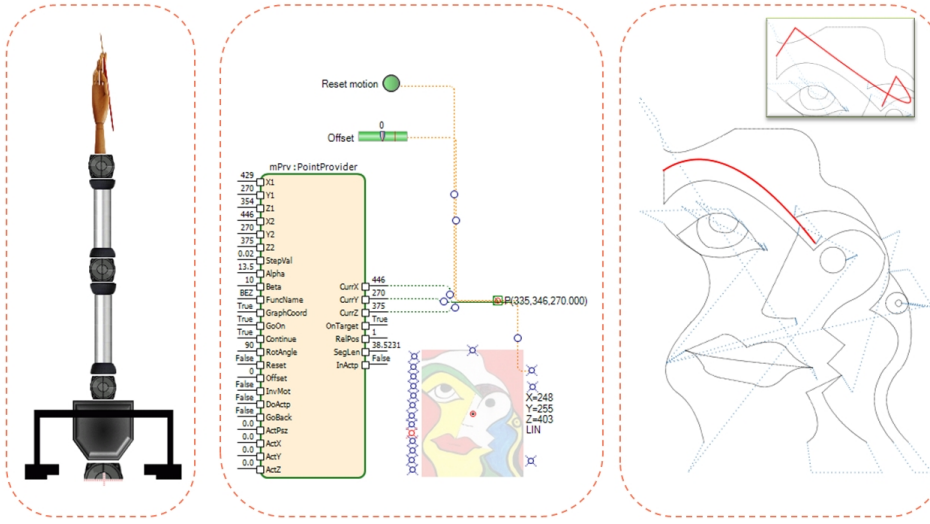


Figure 1. Views on the DSL for robot control

The level of the code generators. An interpreter of MERL-like specifications is implemented [2] for M2T (model to text) transformations, to create code and code generators (generators of generators). Starting from an instance of a model of certain type, the interpreter of MERL-like specifications generates the code concerning the control logic, "meta-logic", and "meta-arithmetic", as well as action reports [4]. Action reports synchronize the state between the model, RTS, and monitoring application, unless the modeling tool is used for monitoring. They are the means for model-level debugging and model execution. The code concerning the "meta-logic" and "meta-arithmetic" is described in Section 5. In brief, this code serves two basic purposes:

1. it increases reliability of control logic by implementing different strategies for recovering the system from an invalid state; and
2. it detects atypical states of the control logic and provides feedback to the modeling tool, which is used to refine the DSL.

The level of the target interpreter or run-time system. The Run-Time System (RTS) is conceived and implemented as an adaptive component that

1. executes both the instructions belonging to a high level of abstraction and binary (native) code;
2. receives and links specification increments without interruption; and
3. simultaneously executes the basic control logic and "meta-logic".

When compared to the similar Programmable Logic Controllers (PLC) used to execute IEC 61131-3 programs [5], it is extended with the concepts from IEC 61499 [6] and a set of libraries for various application domains. In this manner, the modeling of distributed controllers, which is based on finite state automata, is simplified.

Besides Introduction and Conclusion, this paper has six sections. We first give a short description of the architecture of our DVMEx solution for robot control (Section 2). Different approaches to solving the problem of unexpected system states, frequent model variations and their implications regarding the construction of modeling languages are described in Section 3. This section also features a short review of papers related to the RTS-driven approach to the application refinement and the construction of UML profiles for the purpose of model-driven development of industrial process control applications. In Section 4, there is an overview on the evolution of modeling languages, from more general to more domain-specific. We outline different approaches to language construction, from the classification using subtypes to modifications, which are a linguistic concept allowing specification of robot actions at different abstraction levels. In Section 5, we elaborate on ways to specify the "meta-logic" and "meta-arithmetic" at the level of the model, code generator, compiler and run-time system. In Section 6, we describe a platform for model-level debugging, which includes visual tracing. Section 7 contains related work. In Section 8, we conclude this paper by outlining our generic model of DSL refinement and listing our theoretical and practical contributions.

2 THE ARCHITECTURE OF A DVMEX SOLUTION FOR THE ROBOT CONTROL

The architecture of DVMEx Solution for the robot-motion control (Figure 2) represents an extension and a concretization of the basic architecture of DSM solutions [1], which encompasses a DSL, code generator, domain framework, and interpreter or target system. Special attention is devoted to:

1. extending a target interpreter (run-time system), which, in addition to executing complex control operations, executes also **meta-logic** operations and provides feedback to modeling tools;
2. extending code generators, to which we added **action reports**, which synchronize the state of the run-time system with tools for modeling and meta-modeling, and various applications; and
3. using **modifiers** to construct, slice and merge modeling languages.

The meta-modeler creates DSLs using tools for meta-modeling. These languages may be created separately for each domain, i.e., each type of the task that the robot is supposed to execute.

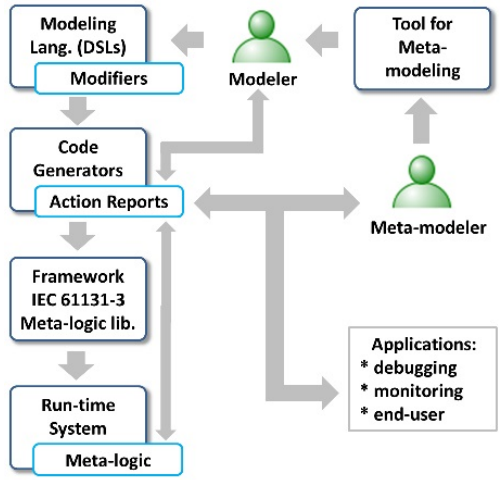


Figure 2. The architecture of the DVMEx solution

There are several groups of code generators. The first one includes generators that generate source code in IEC 61131-3, some GPL, quasi assembly instructions of higher abstraction level or, directly, native code for a particular processor type. The second group includes "generators generating generators", i.e., generators of action reports. They provide the high-level synchronization between the modeling tools, target RTS, and monitoring applications. The third group of generators is used to specify the semantic actions executed during model changes. These generators define rules concerning the integrity of models and transactions, as well as states in which there is synchronization between abstract models and code executed in the RTS. The fourth group includes generators that generate documentation in the PDF or HTML format, either directly or by using document description DSLs as intermediaries.

The framework of the DVMEx solution for robot control includes

1. compilers created to support more thorough specifications of control processes;
2. libraries; and
3. web services.

At the framework level, there are also libraries used to implement complex 3D motions and actions, rules of "meta-logic" and "meta-arithmetic", which is further elaborated on in Section 5.

The run-time system interprets or executes specifications created from abstract models. The available versions support Linux, WinCE and Win XP/7/8/10, from which they utilize memory and file managers, as well as the TCP/IP protocols. The RTS features a preemptive adaptive scheduler managing tasks of different types and priorities, synchronized with the drivers and monitoring applications. Unless the

modeling tool is used for monitoring, client applications are generated using a code generator. In this manner, the action report interpreters, which also support Linux, WinCE and Windows platforms, are used as default applications for the model-level debugging. The visual debugging also uses services of DVDocGen Framework ([7]) to generate PDF or HTML documents. Document scripts, which are specified using a textual DSL for documents, are dynamically generated during the model debugging, providing PDF and HTML documentation for the verification of test cases.

The mechanism concerning the feedback from the RTS to the other levels that are part of the DVMEx architecture is implemented by monitoring the model execution in the RTS and by event triggering. The RTS recognizes predefined (or built-in) states of the interpretation or execution of control logic: Before variable initialization; After variable initialization; Before state changed; After state changed, etc. The monitoring application or modeling tool sets a filter that determines the particular state changes together with the parameter they produce, which are relevant and should be considered. Besides the default states of the RTS, the IEC 61131-3 language and compiler are extended with events, similarly to GPLs. The request for state change is defined as a 4-tuple (Condition, Event_ID, Parameters, RTS_State). These requests may originate from the hardware level, drivers, control logic and meta-logic code.

In addition to these states, which are considered valid, the RTS detects the invalid variable values or wrongly executed arithmetic and logical operations. The strategy for resolving these situations is presented in detail in Section 5. Each of the feedback connections from the lower to the higher abstraction levels provided metadata that are sufficient to make a reference to an instance of the object, relationship, role, model, and code generators that generated certain code portion.

We conclude the overview of the architecture of the DVMEx solution with a remark that it is an extension of DSM architecture. The level of the code generator is extended with action reports, enabling the visual debugging and model-execution without additional programming. The RTS is adaptive and supports updating of the code that is responsible for control logic and meta-logic during run-time.

3 UNEXPECTED STATE OF THE MODELS AND CODE AND MODEL VARIATIONS

An unexpected state represents a paradigm uncovering problems that are related to errors in RTS or in models, model variations, the incompleteness of a modeling language, or the non-adaptivity of the run-time system. There are at least three environmental causes of unexpected states of a system:

1. the lack of appropriate modeling concepts (an incomplete DSL);
2. the lack of code generators; and
3. insufficient power or flexibility of the target RTS that interprets or executes the generated code.

The software for automation and robot control that handles a large number of unexpected states is inefficient and expensive, particularly if general purpose languages or tools are used to create and maintain the software. The appearance of an unexpected state diverts production activities from the expected workflow and decreases the level of their automation. The main characteristics of the unexpected state with respect to the impact on the automation and robot control process are:

- during the execution of a task, an unexpected state cannot be abstracted as any existing model or pattern, described using the existing DSL concepts;
- the actors of a control process are capable of perceiving unexpected states by using their own experience, personal creativity, and the level of knowledge of the modeling framework and language;
- unexpected states are also the ones for which code generators cannot produce the expected code; and
- the control logic of systems in which unexpected states are frequent is most often generated or programmed again, and, during the switch to the new code, the whole control process is temporarily stopped.

A significant amount of research aims at providing support for modeling variant-rich software systems (Software Product Lines) in general. Patterns play an important role in solving the problem of specifying variations. There are a lot of references covering application of patterns in DSM. In [8], the process of creating UML profiles for particular domains is presented. In [9], the authors discussed the role of patterns in the construction of valid DSLs. One of the languages for this purpose is the Common Variability Language (CVL) [10]. CVL is a generic language for modeling variability in models in any DSL based on the Meta Object Facility (MOF). Although conceptually quite comprehensive, CVL still does not offer an adequate support for systematic refinement of models and modeling languages. The following practical constraints limit the use of CVL for this purpose:

- specifying CVL fragments and their referencing need to be more intuitive and thus easier for average users;
- specification of a large number of variations at the level of a model significantly diminishes model understanding and usability;
- specification of variations needs to be provided as a User-Driven Modeling (UDM) activity or influenced by the run-time system, due to the requirement of systematic gathering and classification of variations; and
- a specification of variations needs to be provided not only at the level of models, but also at the level of a target language to which the models are transformed. In practice, this is motivated by the requirement that the effects of a variation may be scoped to different abstraction levels. For example, variation effects may be scoped both to main control logic and monitoring applications.

For the purpose of synchronization we worked out incremental modeling, which includes reliable "on the fly" validation of as many states as possible at the side of the running target system, instead of emulators. We propose improving synchronization that provides:

1. specifications of model variations and synchronization of abstract models, and executable code at the level of meta-model and code generators;
2. systematic refinement of the DSL by means of an analysis system states; and
3. the introduction of the concept of a *modifier*, as means to flexible systematic classification of system states and model variations.

In Figure 3, we present a DSL for modeling of a robot arm. This language, with minor additions, may be used in practice for modeling different types of grippers and industrial robots. This model serves as a running example that illustrates how our approach may alleviate some of the important problems in construction and refinement of DSLs and models, multilevel modeling, model variations and model execution.

The robot arm consists of one or more fingers, each further composed of one or more segments. *FingerSegment* represents the base object type in the DSL for arm. By modifying this type, we created additional types: *Joint*, *Phalangs* and *DistalPhalangs* (see the upper section of Figure 3). Each modification is represented by modifier object, a modification relationship and the roles of the base object and the modified object. Furthermore, the arm also includes *Carpals*, *RootJoint* and *Underarm*. The *ArmAction* object acts as a provider of position, elevation and rotation of finger segments. During language testing, it may be convenient to test only some of its elements or submodels, e.g., only two fingers. The formed relationships with the ArmAction object determine which of the fingers are included in model interpretation. The purpose of such an approach to DSL construction is to modify the metamodel through a model instance. This is achieved by modifying default values, and introducing (or removing) attributes and relationships in order to create new types, supertypes, subtypes and object instances. On each model change, during visual debugging, the following activities are performed:

1. control logic code is generated;
2. a description of the arm's topology is generated; and
3. the model is executed to serve as a visual debugger [11].

Modifiers may be used to express significantly more complex relationships that are used for:

1. multilevel modeling, which combines inheritance and instantiation;
2. inheritance of a type from an instance, a type from a type, an instance from an instance, a type and an instance from the generated code; and
3. specification of a set of allowed model variations.

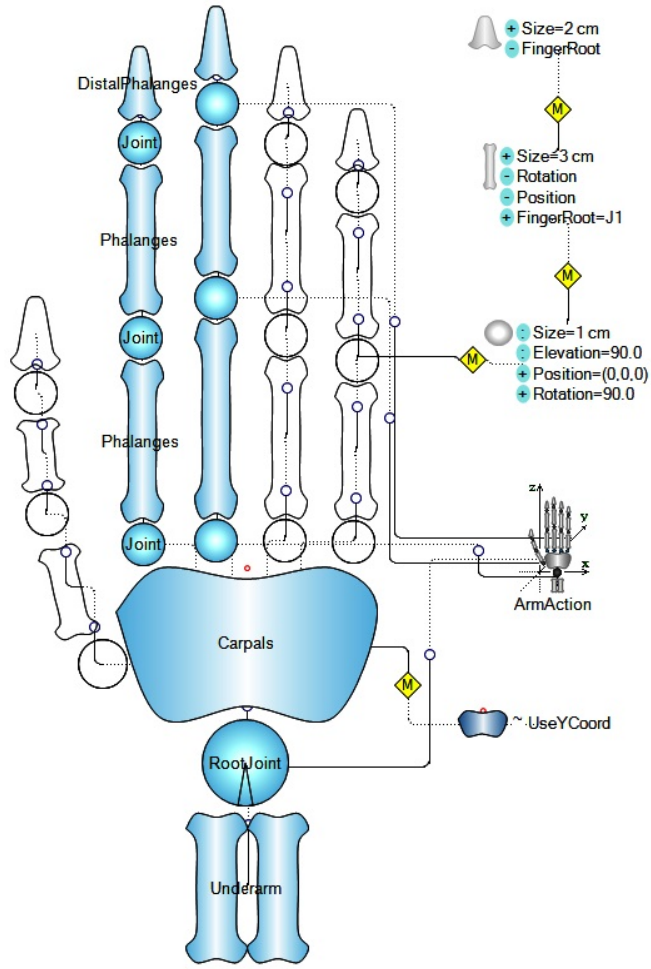


Figure 3. Modifiers as means to DSL refinement

In Figure 3, the *modification* relationship is represented by a diamond shape with the **M** symbol inside. Modified objects are denoted by ⟨*Mod:modName*⟩. The semantic of property modification may be: *instance*, denoted by the (:); *new property*, denoted by the (+); or *inapplicable property*, denoted by the (-) symbol. Within relationships, modifications can change and remove relation and role types. The first two property roles are intuitively clear as they are available with the same meaning in the majority of contemporary modeling languages. The inapplicable property is introduced as a counterpart of the potency concept from the UML extension [12]. However, in our solution it does not require predefining the allowed depth of instan-

tiation and does not restrict any attribute to be applicable again to some subtype or instance.

The example from Figure 3 shows a portion of the DSL, in which the types of robot arms are described. At the most general level, the *FingerSegment* is defined as a segment for which the *elevation* and the *size* are known. For the sake of the completeness of the RMCL specification, a constraint must be used to explicitly express the rule that the *Elevation* depends on the elevation of the previous sequential segments. The *Joint* object is a modification of a segment with the default size of 1 cm and the default elevation of 90 degrees that is also extended with the values for the position of the joint center and the rotation angle (with respect to the X axis in an XY coordinate plane). *Joint* is a subtype of *FingerSegment*. The *Phalanges* object has the default size of 3 cm. However, the rotation angle is not applicable to this object because the corresponding segment may move only along a single axis. The same is true for the position because *Phalanges* is linked to a joint. As it is necessary to know the base joint for each *Phalanges*, we introduced the *FingerRoot* property. Although it may appear that *DistalPhalanges* is a subtype of the *Phalanges* object, that is not the case here. It has the default size of 2 cm and the *FingerRoot* property is redundant because there is at least one joint between *Phalanges* and *DistalPhalanges* that is not the root of the finger.

The presented sequence or hierarchy of object modifications is only one possible case, as variations and their semantics depend on mechanical and topological properties of the robot arm for which various types of programs have to be generated. The given example is an illustration of the problem of frequent variations of models and modeling languages, which may be solved by multilevel modeling, i.e., by integrating modeling and metamodeling.

In order to illustrate such an approach to language construction, we provide an examples of textual representation (the so-called DSL script) of robot arm objects. Each arm specification is a sequence of the DSL scripts given below.

⟨`Joint.Size:1.5 cm,Position(10,20,10)`⟩

The joint of size 1.5 cm at the initial position $(10, 20, 10)$

⟨`Phalanges`⟩

The default Phalanges of size 3 cm

⟨`DistalPhalanges.Size:2.2 cm`⟩

DistalPhalanges of size 2.2 cm

One expected purpose of the RMCL is to simultaneously express different aspects of robot control which may depend on mechanical, thermal, electrical or visual properties. For the purpose of integrating the languages used to model individual aspects, it is necessary to provide a sufficiently flexible mechanism for establishing semantic relationships between language concepts that are related to different aspects. The herein introduced modifiers could be the mechanism for this kind of meta-modeling. The code generators, together with action reports, which act as an interface between the model and the DSL scripts, allow for different interpretations of the examples from Figures 3 and 4.

4 DSL REFINEMENT ON EXAMPLES

The practical benefits of using DSM and modifiers are illustrated in the development of software for the robot that paints. We set up a small DSM team consisting of a software engineer (a DSM specialist), domain expert (a mechanics constructor) and end-user (a painter). The software engineer constructs the language, trying to identify domain-specific concepts from the lower to higher level of abstraction. In this process, the greatest assistance is rapid DSL verification using a set of initial models and a target interpreter, or run-time system supporting an incremental update. The domain expert knows the problems that his or her robot is capable of solving, current and potential user requests, and what kind of robot arm can be made. The greatest assistance is a graphical language and a software tool for quick functional specification, and verification of the application of existing and planned robot models in various environments. The end user, or a person in charge of the application testing, expects to have applications, which without special training, can be used for a variety of robots, and also to automatically document the ability to use them for various tasks.

Figure 4 shows several examples of motions of a robot arm that draws portraits and sketches using a set of curves. Curve parameters are specified by an end user or they are obtained automatically by analyzing the image or 3D objects and recognizing its parts. Figure 4 a) is obtained by combining the basic types of motion in 2D, such as straight lines, arbitrary jumps, circle, sinusoids, impulses, and Bezier curves. The DSL concepts for modeling robot motion logic that allows drawing a portrait on a canvas consists of: a canvas, base type "Motion" and subtypes that match curve types, as well as concepts for describing a robot arm topology. Target framework may be an existing robot control language, but after the construction of the first DSL for the robot motion control, it is already clear that the framework should be also improved and simplified. In our case, the DVRobCon Motion Framework reduces all the motions to the finite set of curves with properties: the start and end point in space, curve type, amplitude height, number of impulses or oscillations, rotation angle, and curve offset. With such a framework, model objects representing motions can be mapped one-to-one into motion framework or library. When a DSL should include concepts such as eye, nose, eyelash, such instances from a model are transformed into $1..M$ commands of the motion framework.

Figure 4 b) shows the motion where a user at the time of motion sends a signal or a command to the robot to draw the straight line of a certain length, relative to the current point, and then return and continue the motion. Such a request is solved by introducing a new language concept named **Deviation Point**, which is not a true subtype of the basic motion types, because it also contains an event (triggering an operation). If a robot arm uses a brush instead of a pencil or pen, some of the lines can be drawn with different thicknesses – thinner to the ends, and thicker in the middle (Figure 4 c)). We introduce a new DSL concept named **EyeLash**, as a modification of the Bezier curve. Considering the domain expert

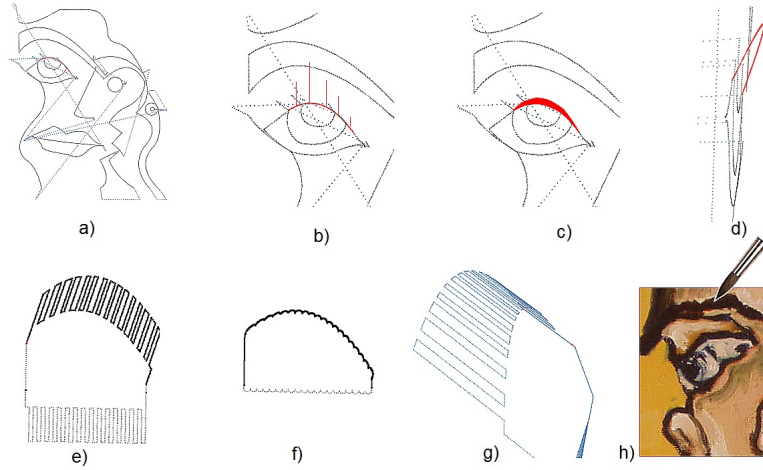


Figure 4. Refinement of robot motion control language

experience (painter) the line from Figure 4 c) can be obtained by setting rotation angle to 70 degrees and line (brush) offset to 5 mm (Figure 4 d)).

Further steps in refinement of the DSL for painting bring a modification of Bezier curve, which contains impulses (Figure 4 e)). In the electronics terms, this is the modulation of a Bezier curve with impulses. The curve performing modulation is called a **modulation pattern**. Therefore, the DSL is extended with the concept **Motion Modulator**. This modulator is a curve that affects one or a group of curves, and depending on the orientation in the space, it produces new, complex or composite motions. Although it is mostly clear, from the mathematical point of view, how to implement such a path, things coalesce when it is necessary to ensure continuous motion of uniform speed, accelerated, or motions which are re-modulated at the run time. Such modulators, even the simplest ones, solve the problems of performing complex 3D operations in CNC machines. These modulations give us the freedom to decide about the portrait while painting, not in advance. In Figures 4 e) and 4 f), modulation patterns are shown in the bottom, while in the top it is shown resulting modulated Bezier curve for EyeLash.

The extension of the DSL for painting from 2D to 3D, i.e., from the DSL for painting to the DSL for sculpting, is presented in Figure 4 g). A modern artist desires wired lashes bent in a pulse-like shape. We introduce a new DSL concept named **3D EyeLashes**, whose shape, dimensions, inclination, thickness, etc., are parameterized.

When it comes to a robot as a painter and Domain-specific Modeling, in the process of refining DSLs, the goal is to create a set of modeling concepts that express important characteristics of

1. painting epochs and directions,

2. techniques and materials, as well as
3. individual characteristics of the painter.

We are convinced that objectives 1 and 2 can be realized to a significant extent with at most two levels of modulation, i.e., modification of the base curves at the time of painting or sculpting. When it comes to the construction of painter-specific language concepts, modulation patterns should also express the motoric, intellectual and emotional properties of painters or sculptors. Such patterns are formed both in advance and at the time of work on a particular painting or sculpture.

Refining the DSL with concepts such as EyeLash that reflects the characteristics of a painting direction or a painter also affects the objects of the control logic and painting environment. Figure 5 shows the modifiers for identifying the subtype of the function block that calculates the points in 3D space (libID), the brush size (disCarpSize), the position of the canvas, the rotation center, and the set of curves for drawing EyeLash (motAndPos). All individual modifiers are grouped into EyeLash, as modification type. Default specifications of user applications are generated in a human-readable XML format for interpreting under different operating systems and hardware platforms (Listing 1). The specification contains several parts:

- submodels, or forms and subforms;
- visual properties of form elements in applications;
- commands for communication with the target RTS, and their invocation rules (cyclic or upon event occurrence);
- commands for the exchanging properties or events between the form elements; and
- list of modifiers with an identifier of the group they belong to.

ModelModifiers contain object modifications related to control logic, arm topology, and objects in the environment, as well as their different layouts. During the interpretation, an end user can select a group or individual modifier, to select view and associated layout. The end user is also allowed to change properties in the execution time. The DSL semantics cannot be changed through user applications, but some properties, having influence on default values, can be changed. Updated properties are taken from the modeling tool in run-time, and serve as the basis for updating default property values or domain definitions. Beside meta-logic and meta-arithmetic, which are described in the next session, feedback gained from the target interpreter of applications is also used for automated refinement. The software architect and end-users simultaneously "debug" the model and the generated code. In the scenario of model execution, or visual debugging, after each change in the model, the increment of program code for the robot controller, user applications and action reports are regenerated. The time elapsed from the change in a model to the new start of control logic and user applications is a few seconds. In most cases, this is also the time spent for demonstrating the code and application validity.

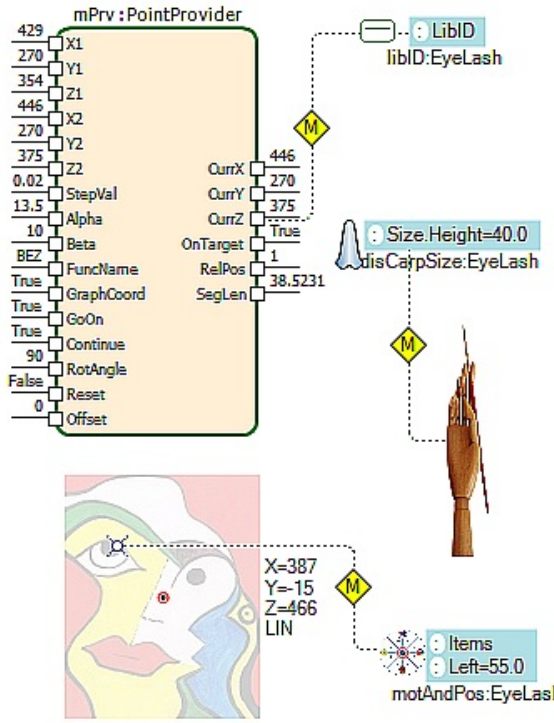


Figure 5. Representing and grouping of modifiers

5 META-LOGIC AND META-ARITHMETIC

By meta-logic and meta-arithmetic we denote a set of operations that provide additional information about the context of the execution of logical and arithmetic operations. In short, meta-logic includes tracking the status of logical operations, while meta-arithmetic includes tracking the status of arithmetic operations. Both operation types calculate the logical value regarding the correctness of operations during execution and, by applying different strategies, may help make the control process as stable as possible. Herein, both operation types are often referred to as meta-logic. The DVMEx approach allows for the definition of meta-logic at each of the levels forming the architecture of the DSM solution:

1. at the level of the meta-model and model, by means of language concepts, by specifying the semantic domains of attributes and model constraints;
2. at the level of code generator, where code is generated for the rules of meta-logic, as well as when rules are not explicitly expressed by the model;

```
<Submodels>
  <Submodel name="Control logic",...>
    <SubmodelElems>
      <Element name="libID" Value="MotLib_2",.../>
    </SubmodelElems>
  </Submodel>
</Submodels>
<Application>
  <Object name="Canvas" Left="60.0", ... />
</Application>
<RTS_Commands>
  <Events>
    <event object="DVRobCon.doMotions" name="OnSetValue">
      <![CDATA[DB SV "DVRobCon.doMotions","(())"]]>
    </event>
  </Events>
  <Cyclic time="200">
    <cmd condition="CycleID=1" action="DB GV (varList)"]]></cmd>
   </Cyclic>
  <DirectMappings>
      <mapping>
        <![CDATA[:.motionSpeed.SendToRTS(Value);]]>
      </mapping>
  </DirectMappings>
</RTS_Commands>
<ModelModifiers>
  <ModelModifier name="motAndPos" GroupID="EyeLash">
    <Object name="Canvas">
      <update_prop Items="..." />
      <update_prop Left="55.0" />
    </Object>
  </ModelModifier>
</ModelModifiers>
```

Listing 1. Specification of an end-user application

3. at the level of the target language compilers (in this case an IEC 61131-3), by using properties that define meta-logic for certain types of data and operations; and
4. at the level of the RTS, by defining filters that determine the detection and reporting rules about the operation status and unexpected states.

In Figure 6 there is an example of meta-logic that is explicitly specified by the model. The rounded upper right section of the figure illustrates the meta-model, i.e., the modifier type, while the central section of the figure is devoted to the model instance. The instance of the function block div2: DIV, which uses division

```
if Type='DIV' then
        'VAR
                validate1:VALIDATE;
        END_VAR'
        /* Code for DIV(slider_1,slider_2) */
        validate1:= VALIDATE(div2.OUT,SDef);'
endif
```

Listing 2. An example of program code for validation

to calculate the length of the robot step based on the distance to some object (sliders to the left), produces the undefined value. The domain-specific function block VALIDATE determines the length of the step based on the input value shown (produced) by the scale `Sdef` to the left. In order to resolve the unexpected state of control logic, which is also invalid in this case, function block VALIDATE is used.

In relation to the discussion of the modeling approaches in the previous section, the approach is most similar to the modeling using modifiers. This kind of a DSL is to a great extent an example of a modeling language (RMCL), which is often used in practice. The textual representation (DSL script) of the model featured in Figure 6 may be of the form ⟨`DIV.validate`⟩`valRepl_Val`.

In case the modeling language does not feature VALIDATE, but that kind of a function block is available in the target language or library, the same meta-logic may be specified using a code generator as presented in Listing 2.

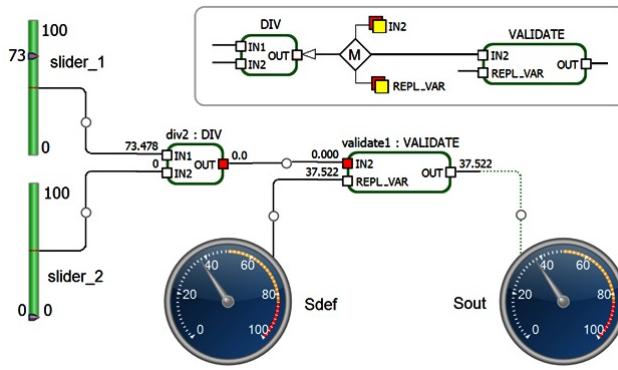


Figure 6. Meta-logic at the level of the meta-model and model

In case function block VALIDATE is not available, the native code that checks the input parameters IN1 and IN2 for DIV is generated. When a compiler is used to implement the meta-logic, it generates additional native code instructions. The native code checks the processor registers to determine the operation status (e.g., overflow, underflow, and divByZero) and assigns the status to a temporary result or

variable. Native code for the meta-logic may be of the form shown below. In order to be more readable, Assembly for Intel x86 processors is given in Listing 3. instead. The assembly code sets status for a variable named DIV2. If any of variables used for calculating is invalid, then the variable status will be also invalid.

Irrespective of the level at which the code for meta-logic is provided, we defined several strategies concerning the evaluation of meta-logic expressions (Figure 7):

1. *propagate* – where operation statuses are propagated to all subsequent operations in which some variable or temporary result is used;
2. *reset_assign* – where the status is reset to valid whenever a valid value is assigned to some variable;
3. *reset_cycle* – where the statuses of variables are reset before each new program execution cycle; and
4. *ignore* – where meta-logic is not executed, i.e., operation statuses are not tracked.

In the example in Figure 7, a function block for the data type conversion is used, from a long to real value, but the input lreal value 5e+38 is greater than the maximum real value. The type convertor is marked by `tc:lreal_to_real`, and the symbol above which there is only `tc` is a graphical representation for assigning the constant, which changes the output variable.

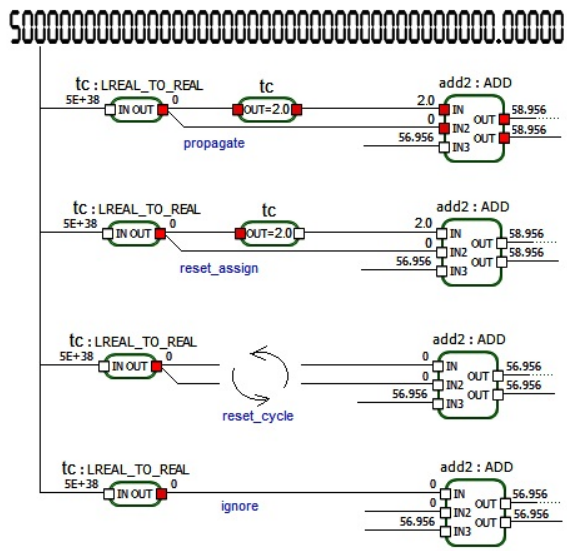


Figure 7. Different strategies concerning the meta-logic

The RTS allows for incremental updating of the code and dynamic change of the meta-logic strategy. For the reliable functioning of control logic, the most suitable

```
mov DIV2$status, 1
mov al, __SLIDER1$status
cmp al, 0
jne ok3
mov __DIV2$status, 0
ok3:
movsx ebx, WORD PTR __SLIDER1
mov al, __ SLIDER2$status
cmp al, 0
jne ok4
mov __DIV2$status, 0
ok4:
movsx ecx, WORD PTR __SLIDER2
cmp ecx, 0
jne ok5
mov __DIV2$status, 0
jmp skip6
ok5:
mov eax, ebx
cdq
idiv ecx
mov ebx, eax
mov __DIV2, bx
```

Listing 3. An example of generated Assembly code

strategy is *reset_assign*. For the testing of control logic and model-level debugging, the most suitable strategy is *propagate*. For the detection of deviations in the status of control logic between the cycles of program execution, the most suitable strategy is *reset_cycle*. The *ignore* strategy is used in well-checked control programs, which are automatically generated from well-checked models using well-checked modeling languages.

6 PLATFORM FOR DSL MERGING AND MODEL-LEVEL DEBUGGING

The efficacy of the construction, testing and application of new DSLs depends on swift and simple utilization and adaption of existing languages (DSL reusability), patterns and target systems that execute specifications. In the context of the theoretical discussion concerning syntax and semantics merging [13], we present our practical solution – a platform that supports model merging and model-level debugging which utilizes merged languages.

For the purpose of modeling in robotics and automation we use three DSLs, which were merged by meta-model integration. In the example from Figure 8, the first DSL (RMCL) is used to model topological properties of arm, foot or body, their

motions and actions. The constructs from this DSL are shown in the form of a blue hand. The positions of segments, with the exception of *RootJoint*, are not relevant. However, relationships between segments do matter, as they determine topological characteristics of the hand. The particular layout of the segments was chosen for its more appealing look. For new cases, the construction of a DSL from the group requires several days.

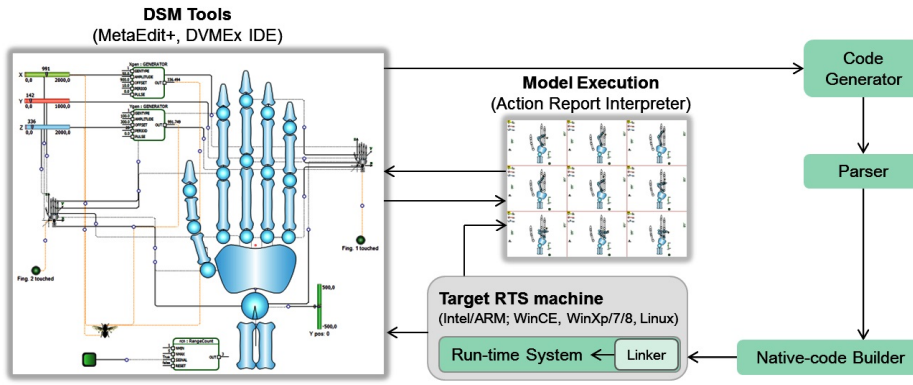


Figure 8. Platform for model execution and model-level debugging

The second DSL is a language for function blocks diagrams. The elements of the language (named FBD DSL) are depicted as green rounded rectangles with input and output ports. This language contains constructs of the target framework according to the IEC 61131-3 specification. It is also extensible and acts as an interface to an auto-adaptive run-time system. Code generators that are specifically written for FBD DSL are generic and applicable to the merged language. FBD DSL supports using a model to describe control logic to the greatest detail (the level of variables and operations), instead of making it part of the code generator. The relationships between objects of different DLSs are expressed by mapping the properties of RMCL objects to the ports of function blocks in FBD DSL and vice versa. The complexity of the model may arise as a result of the presence of objects from different DSLs and redefinitions of linguistic concepts in the same instance (Figure 8, left part). However, this may be resolved through the DSM tool by utilizing different model views (application, control-logic, topological, domain-specific) and applying decomposition.

The third DSL features linguistic concepts that describe objects and properties of the environment in which the robot operates. It is constructed by modifying a set of general-purpose linguistic concepts, such as analog and digital controllers, sensors, scales, switches, sliders, displays, etc. In the previous figure, the elements of this language include sliders in the upper left section and green switches. By using the objects of this DSL, it is possible to simply construct virtual signals and generate controller drivers [11]. This method of generating “domain-specific drivers”

from models is beneficial as it leads to better utilization of hardware resources and faster native code, which is comparable to the optimized code provided by a C++ compiler. This DSL is part of the development environment and, therefore, it does not require additional time for its construction. Any additional modification requires at most one day of work.

Visual debugging is a process of executing models that is performed in parallel with editing "on hot" without stopping the execution of the current program within the RTS. On model change, the generated code may be forwarded to the parser or the native code may be directly generated. The target RTS uses a dynamic linker to receive specification increments and links the control logic code to variables. Changes are performed within transactions. The completion of a transaction is reported to the modeling tool by the RTS. Visual tracing is achieved by MERL-like generators (action reports), which change model state within the DSM Tool based on the state of interpretation or program execution. The average time between a model change and the start of the execution of new native code is approximately 200 ms.

7 RELATED WORK

Software engineers demand significantly improved methodologies and tools to develop and maintain reliable robotic applications. Robotic systems are inherently complex, and developers must integrate various software tools and electronics from different manufacturers ([14]). Recently significant efforts have been invested into the research of model-based approaches and tools aimed to facilitate software development in robotics ([15, 16, 17, 18, 19]). Also, numerous successful applications of DSLs are reported in various domains including robotics [20]. Our research is directed toward further evolution of the DSM approach with model execution and modifiers. The model-execution concept has recently attracted significant attention from the academic community. It may be found under different names, such as live programing [21]. Model execution proved to be a powerful means for the dynamic validation and verification of models. We implemented this concept using action reports that are extended with a set of commands for communication with the target RTS [22]. Also, we introduced modifiers and illustrated their use in several examples. In the context of graphical DSLs, modifiers are means to simple and intuitive merging and customization of languages, as well as language application to new problems.

In [23], Hästbacka et al. describe the application of the MDD and DSM approaches to the development of industrial process control applications. Their approach is based on the utilization of the UML Automation Profile modeling concepts (UML AP). Based on our experience, there are several reasons why the application of this approach to the generation of process control applications is limited. First, both UML and the construction of UML profiles are complicated and, to the average user in the domain of automation and process control, they do not offer a faster re-

sponse to requests or a better insight into the system being modeled. Second, OPC interfaces are not abstract in the manner that they could be described using a simple language. For that reason, the high-level synchronization between the models and the generated process control applications is limited. Third, the approach described by Hästbacka et al. does not provide sufficient attention to run-time systems as interpreters of models that solve numerous shortcomings in the generation of code for GPL compilers, instead of doing that for domain-specific RTSs.

Song et al. [24] introduce an approach for connecting architectural models with run-time systems with bidirectional transformation and automated changing of architectural models according to changes in the run-time system. When compared to the approach of Song et al., our approach does not use architectural but domain specific models and, as a result, it may be suited to a wide variety of audiences, not only software architects. Furthermore, in our approach the implementation and its connection to models are automatically generated, which adds more reliability to the implementation of the approach. Moreover, the approach of Sung et al. allows only changes in the model that can be captured in the architectural language metamodel, while our approach also allows changes in the metamodel, i.e., introduction of new concepts for modeling systems. Finally, our approach also enables incremental changes of models and run-time systems during execution including changes in which novel modeling elements are introduced.

There is an analogy between the two seemingly unrelated domains of application: document engineering and robot-motion control. In both domains, the efficacy of modeling tools depends on their support for merging languages that model different dimensions/aspects of documents or robot control. A document is a multidimensional entity featuring the following dimensions: content, layout, structure, role in a real system and states. In some cases, it may not be possible to generate valid code for control logic, e.g., during robot motion modeling, when a robot step cannot be related to foot shape and other objects in the environment where the robot is moving. However, by using solutions for syntax and semantics merging of DSLs, we may simplify parallel viewing and modeling of different document dimensions or aspects of robot usage. In [25], the authors present some preconditions, as well as problems in merging syntax and semantics, and meta-model inheriting and slicing. On the other hand, our discussion and contributions are more of a practical nature. We introduced modifiers and illustrated their use in several examples. In the context of graphical DSLs, modifiers are means to simple and intuitive merging and customization of languages, as well as language application to new problems.

Another topic of related research is about multilevel modeling. The most recent advances in this field are related to the concept of deep instantiation, supported by DeepJava. The authors in [26, 12] propose a way for avoiding the shortcomings of programming languages that result from their two-level architecture, seen as type–instances paradigm. The proposed concept of deep instantiation is not applicable to the description of model variations because of the following three reasons:

1. it requires the instantiation depth to be specified in advance;
2. it does not support the relationships in which an attribute from the supertype may be removed from the subtype; and
3. it does not support meta-data-based inheritance of the type from an instance.

8 CONCLUSIONS

In model-driven software development (MDSD), meta-model refinement is an activity which improves the current expressiveness of a language, i.e., improves the capacity of the language to precisely express the properties of the real system being modeled. The DVMEx approach provides the means to refine each level of the DSM solution: the DSL, the code generator, the framework, and the RTS. The synchronization between the tools used separately for each level may provide good productivity and validity in the DSL refinement. In most of the existing tools, the synchronization between the levels is unidirectional.

In Figure 9 we outline the process of refinement. The left column containing the ellipses denotes standard activities that are related to DSM. In the right column, there are activities that are specific for the DVMEx approach. At the higher abstraction level, it is the modifier construction, which acts as means to flexible evolutionary extension of the language semantics. The second level is the usage of modifiers. Underneath that level, there is the execution of models or automatically generated applications. For each model, at least one default client application for model debugging and monitoring the model execution is generated. Since arbitrary controls may be used for monitoring by mapping the DSL concepts to user control properties during execution, model variations may be tested very fast. Modifiers may also be used at the level of the code generator. In this manner, there is a temporary solution for problems arising from the imprecisely specified hierarchy of DSL objects or imprecisely described relationships or object roles. During execution, the RTS detects unexpected states. Some of them may be trivial, such as division by zero, but they may also include states that cannot be recognized in the predefined state space of the control logic program.

For the purpose of developing intelligent controllers for robots, we have devised an approach that is based on the DSM approach. We significantly improved each of the levels of the DSM architecture. For the purpose of verifying the DVMEx approach, we created the tools that, together with MetaEdit+, may be used to verify the reliability, flexibility, speed, and the simplicity of the integration of the control logic into an arbitrarily complex real system. In the rest of Conclusion, we list the theoretical and practical contributions, whose application may improve the development of intelligent robot controllers and the development of measurement and control systems in general.

Contributions to the theory. The RTS is conceived as an adaptive system with dynamic scheduling and linking of the code of control logic and meta-logic.

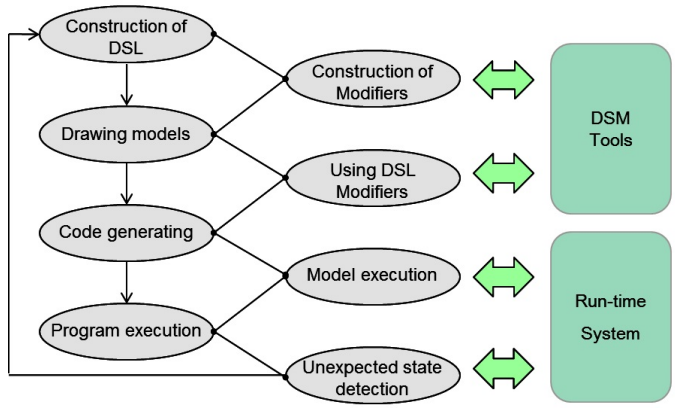


Figure 9. A generic model of DSL refinement

Changes in the rules of scheduling of control tasks in run time, as well as dynamic mapping of the signals of a real system to the variables of the control code allow for full flexibility when changing the purpose and behavior of the robot, even for the states that have not been programmed in advance. The generators are simultaneously used for code generation, debugging, execution and monitoring of the control processes. At the level of the code generator, modifiers may be described, as model or submodel variations, whose purpose is systematic collection and processing of knowledge for the refinement of the DSLs. We extended the IEC 61131-3 with concepts for the description for finite state machines and message exchange protocols, which simplifies the implementation of the event-driven control logic. At the level of DSL construction and usage, we define an approach of evolutionary refinement with modeling variations. We introduced the modifier concept, which integrates inheritance, instantiation and supports multilevel modeling.

Contributions to the practice. We made an auto-adaptive RTS for ARM and Intel hardware platforms. It is applicable to problems involving simple control logic in embedded systems, as well as complex problems, such as automatization of the whole production process or the control of intelligent human, bird or snake-like robots, in which the RTS interprets complex motion models, models of energy consumption and models for the recognition of the signal of a real system.

The IEC 61131-3 compiler with different variants of meta-logic offers a significant advantage over the existing GPL compilers, where the logic has to be embedded into source code. The code generator level provides synchronization of RTS, modeling tools and monitoring applications without the need for programming. This is achieved using the code generators that generate code generators. Despite the fact that nesting a language in another language may be bad because of the low-

ered readability, this approach to the generation of monitoring applications provides good productivity in software development. The issue of the lower readability of the generators that generate generators is straightforwardly overcome within tools that support the construction of DSLs (meta-modeling) using a graphical interface. The semantics of such generators is described at the level of the meta-model. All components that are part of the DVMEx solution are simply integrated into various tools for meta-modeling and modeling.

In addition to software engineering, validity of the approach, devoted to handling variation of products and DSL models describing these products, has been demonstrated in electronics, and partly mechanics. We made several usable prototypes of controllers for managing robots with sensors, for which drivers and virtual signals are also generated from the model. Also, we have developed several variants of pneumo-mechanical grippers for robots in the textile industry, which perform complex operations in a confined space. Model-level debugging is not limited to generating and debugging the control logic and user applications program code, but it is also applicable to debugging in the electronics and mechanics domain.

Our further research work is aimed at extending the DVMEx approach to the needs of the development of intelligent robots of various shapes and purposes, but primarily for those that need to learn the command language during task execution and provide information about the need to refine these languages.

Acknowledgement

The authors would like to kindly thank Dr. Juha-Pekka Tolvanen from the University of Jyväskylä for his valuable support and proof reading.

REFERENCES

[1] Kelly, S.—Tolvanen, J.-P.: Domain-Specific Modeling: Enabling Full Code Generation. Wiley-IEEE Computer Society Press, 2008. ISBN: 978-0-470-03666-2.

[2] MetaEdit+ Workbench, Workbench User's Guide.

[3] Robot Operating System (ROS). Availaible at: `http://wiki.ros.org/`.

[4] Djukić, V.—Luković, I.—Popović, A.—Ivančević, V.: Model Execution: An Approach Based on Extending Domain-Specific Modeling with Action Reports. Computer Science and Information Systems (ComSIS), Vol. 10, 2013, No. 4, pp. 1585–1620, doi: 10.2298/CSIS121228059D. ISSN: 1820-0214.

[5] International Standard IEC 61131-3: Programmable Controllers – Part 3: Programming Languages. International Electrotechnical Commission, 2003.

[6] International Standard IEC 61499: Function Blocks, Part 1–Part 4. International Electrotechnical Commission, 2005.

[7] Djukić, V.: DVDocGen Framework, Application Interface. 2009, 88 pp. Availaible at: `http://www.dvdocgen.com/Framework/DVDocFramework.pdf`.

[8] Kim, D.-K.—France, R.—Ghosh, S.: A UML-Based Language for Specifying Domain-Specific Patterns. Journal of Visual Languages and Computing, Vol. 15, 2004, No. 3-4, pp. 265–289, doi: 10.1016/j.jvlc.2004.01.004.

[9] Schaefer, C.—Kuhn, T.—Trapp, M.: A Pattern-Based Approach to DSL Development. SPLASH'11, Workshop on DSM'11, 2011, pp. 39-46, doi: 10.1145/2095050.2095058.

[10] Common Variability Language (CVL), CVL 1.2 User Guide. Availaible at: `http://www.omgwiki.org/variability/doku.php`.

[11] Djukić, V.: Various Demos of DSL Construction, Application and Refinement in Robotics, Automation and Design of Medical Devices. Availaible at: `https://www.youtube.com/channel/UCqyYnYD6J5fEeb6Ni3YLuKg`.

[12] Kühne, T.—Schreiber, D.: Can Programming Be Liberated from the Two-Level Style: Multi-Level Programming with DeepJava. Proceedings of the 22nd Annual ACM SIGPLAN Conference on Object-Oriented Programming Systems and Applications (OOPSLA'07), 2007, pp. 229–244, doi: 10.1145/1297027.1297044.

[13] Degueule, T.—Combemale, B.—Blouin, A.—Barais, O.—Jézéquel, J. M.: Melange: A Meta-Language for Modular and Reusable Development of DSLs. Proceedings of the 2015 ACM SIGPLAN International Conference on Software Language Engineering (SLE 2015), 2015, pp. 25–36, doi: 10.1145/2814251.2814252. ISBN: 978-1-4503-3686-4.

[14] Djukić, V.—Popović, A.—Tolvanen, J.-P.: Domain-Specific Modeling for Robotics – from Language Construction to Ready-Made Controllers and End-User Applications. Proceedings of the 3rd Workshop on Model-Driven Robot Software Engineering (MORSE'16), Leipzig, Germany, ACM, 2016, pp. 47–54, doi: 10.1145/3022099.3022106. ISBN: 978-1-4503-4259-9.

[15] Trojanek, P.: Model-Driven Engineering Approach to Design and Implementation of Robot Control System. 2nd International Workshop on Domain-Specific Languages and Models for ROBotic Systems (DSLRob'11), 2011.

[16] Piechnick, C.—Götz, S.—Schöne, R.—Assmann, U.: Model-Driven Multi-Quality Auto-Tuning of Robotic Applications. Proceedings of the 2015 Joint MORSE/VAO Workshop on Model-Driven Robot Software Engineering and View-Based Software-Engineering, L'Aquila, Italy, ACM, 2015, pp. 35–40, doi: 10.1145/2802059.2802063.

[17] Adam, K.—Butting, A.—Heim, R.—Kautz, O.—Rumpe, B.—Wortmann, A.: Model-Driven Separation of Concerns for Service Robotics. Proceedings of the International Workshop on Domain-Specific Modeling (DSM 2016), Amsterdam, Netherlands, ACM, 2016, pp. 22–27, doi: 10.1145/3023147.3023151. ISBN: 978-1-4503-4894-2.

[18] Pradhan, S. M.—Dubey, A.—Gokhale, A. S.—Lehofer, M.: CHARIOT: A Domain Specific Language for Extensible Cyber-Physical Systems. Proceedings of the Workshop on Domain-Specific Modeling (DSM 2015), SPLASH'15, Pittsburgh, USA, ACM, 2015, pp. 9–16, doi: 10.1145/2846696.2846708. ISBN: 978-1-4503-3903-2.

[19] Saglietti, F.—Meitner, M.: Model-Driven Structural and Statistical Testing of Robot Cooperation and Reconfiguration. Proceedings of the 3rd Workshop on Model-Driven Robot Software Engineering (MORSE'16), Leipzig, Germany, ACM, 2016, pp. 17–23, doi: 10.1145/3022099.3022102. ISBN: 978-1-4503-4259-9.

[20] Nordmann, A.—Hochgeschwender, N.—Wrede, S.: A Survey on Domain-Specific Languages in Robotics, Simulation, Modeling, and Programming for Autonomous Robots. In: Brugali, D., Broenink, J. F., Kroeger, T., MacDonald, B. A. (Eds.): Simulation, Modeling, and Programming for Autonomous Robots (SIMPAR 2014). Springer, Cham, Lecture Notes in Computer Science, Vol. 8810, 2014, pp. 20–23, doi: 10.1007/978-3-319-11900-7_17.

[21] van Rozen, R.—van der Storm, T.: Model Execution: Toward Live Domain-Specific Languages: From Text Differencing to Adapting Models at Run Time. Software and Systems Modeling, Vol. 18, 2019, No. 1, pp. 195–212, doi: 10.1007/s10270-017-0608-7. ISSN: 1619-1374.

[22] Djukić, V.—Popović, A.—Lu, Z.: Run-Time Code Generators for Model-Level Debugging in Domain-Specific Modeling. Proceedings of the International Workshop on Domain-Specific Modeling (DSM 2016), Amsterdam, Netherlands, ACM, 2016, pp. 1–7, doi: 10.1145/3023147.3023148. ISBN: 978-1-4503-4894-2.

[23] Hästbacka, D.—Vepsäläinen, T.—Kuikka, S.: Model-Driven Development of Industrial Process Control Applications. The Journal of Systems and Software, Vol. 84, 2011, No. 7, pp. 1100–1113, doi: 10.1016/j.jss.2011.01.063.

[24] Song, H.—Huang, G.—Chauvel, F.—Xiong, Y.—Hu, Z.—Sun Y.—Mei, H.: Supporting Runtime Software Architecture: A Bidirectional-Transformation-Based Approach. Journal of Systems and Software, Vol. 84, 2011, No. 5, pp. 711–723, doi: 10.1016/j.jss.2010.12.009.

[25] Tolvanen, J.-P.—Kelly, S.: Integrating Models with Domain-Specific Modeling Languages. Proceedings of the 10th Workshop on Domain-Specific Modeling (SM'10), Reno, Nevada, USA, 2010, Art. No. 10, doi: 10.1145/2060329.2060354.

[26] Atkinson, C.—Kühne, T.: The Essence of Multilevel Metamodeling. In: Gogolla, M., Kobryn, C. (Eds.): UML 2001 – The Unified Modeling Language. Modeling Languages, Concepts, and Tools (UML 2001). Springer, Berlin, Heidelberg, Lecture Notes in Computer Science, Vol. 2185, 2001, pp. 19–33, doi: 10.1007/3-540-45441-1_3.

Verislav Djukić is Software Architect employed in Djukic Software GmbH, Germany. He is involved in the development of modeling tools for robotics and automation and PLC based runtime systems for the purpose of model execution. He holds his Ph.D. in software engineering from the University of Novi Sad, Serbia. His current research interests are related to the handling of model variations in DSM tools and construction of robot motion framework in humanoid robot painting.

Aleksandar Popović graduated from the Faculty of Science at the University of Montenegro. He completed his Master's degree (2 year) at the University of Novi Sad, Faculty of Technical Sciences. He received his Ph.D. in computer science from the University of Montenegro in 2013. He is Assistant Professor at the Computer Science Department of the Faculty of Science and Mathematics, University of Montenegro. His current research interest includes domain-specific languages and domain-specific modeling. Also, he has been actively involved in the development of DSLs for embedded and real-time systems.

Ivan Luković received his graduate diploma degree (5 years) in Informatics from the Faculty of Military and Technical Sciences in Zagreb in 1990. He completed his Masters's degree (2 year) at the University of Belgrade, Faculty of Electrical Engineering in 1993, and his Ph.D. at the University of Novi Sad, Faculty of Technical Sciences in 1996. Currently, he is Full Professor at the Faculty of Technical Sciences of the University of Novi Sad, where he is Lecturer in several Computer Science and Informatics courses. He is the head of B.Sc. and M.Sc. study programs in Information Engineering – Data Science. His research interests are related to database systems, business intelligence systems, and software engineering. He is the author or co-author of over 150 papers, 4 books, and 30 industry projects and software solutions in the area.

Vladimir Ivančević is Assistant Professor in applied computer science and informatics at the Faculty of Technical Sciences, University of Novi Sad, Serbia. His main research areas include data science, databases, and information systems. He has participated in diverse research projects involving application of computer science and informatics in education, public health and epidemiology, and software engineering.