A Study of the Parameters Concerning Load Balancing Algorithms

Ioannis Psoroulas, Ioannis Anagnostopoulos, Vassili Loumos and Eleftherios Kayafas

National Technical University of Athens,
School of Electrical & Computer Engineering
Department of Communications, Electronics and Information Systems
Zographou Campus
15773 Athens, Greece

Summary
Computer system performance depends on load balancing which should concern communication delay, grid topology, workload and the negotiation protocol. The interactions and interdependences between these factors and their correlation with the selected load balancing algorithms are studied in the current paper. Vital issues are considered and extensively examined through the systematic introspection and the comparison of two load balancing algorithms, a static and a dynamic one. The former is the well-known deterministic Round-Robin, whereas the latter has been developed for the needs of our research. In addition, a flexible simulation framework is implemented where the experiments take place. Appropriate metrics are formulated so that their joint examination reveals the behaviour of the system in terms of performance. Performance degradation might result when high information policy complexity is combined with important communication overheads. However, system efficiency can be improved when intense workload is adequately combined with increased delay. Precision of the system's state information is always compensated by the simplicity of the negotiation protocol. Moreover, the grid's topology is examined. Equations, which reveal the dependency of performance and topology, are derived, through the methodical analysis of potential load balancing scenarios and they are confirmed by the experimental results.

Key words:
Load balancing algorithms, Computational Grid, discrete event simulation

1. Introduction

Several load balancing algorithms have been developed aiming to improve the performance of a computing system. Their complexity varies. Some of them are inspired or conceptually designed according to physical disciplines [1], [2]. Others are based on adaptive [3], [4] and dynamic policies [5] or even apply innovative methods such as partitioning [6]; whereas there are some which are simple [7] but effective under particular conditions.

Our consideration is focused on the description of main factors, such as communication delay, topology, workload, and the complexity of the negotiation protocol [8] (in this study the negotiation protocol includes all the exchanged messages), which strongly affect the system's performance and consequently determine the issues that an algorithm should have the capacity for. In the respective research field, numerous studies have already been published. For example, Mirchandaney et al. [9] analyze the effects of the delay on the performance of three algorithms formulating theoretic models, and Banawan et. al [10] verify its significant impact. Zhou [11] simulates seven algorithms and compared their performance in respect of the applied workload and other vital factors. Eager et al. [12] study the appropriate level of complexity for load sharing policies.

Our conclusive results are based on the examination and the comparison of two algorithms; the classic static Round-Robin and a dynamic one, the Live Update Information Algorithm (LUIA) which is accordingly framed for the current research. We have implemented a flexible framework where our experiments are performed. The framework is based on an existing discrete event simulation platform and supports additional simulation scenarios since its abstract design and its extensive external configuration, through an XML file, allows the user to form in a simple way basic models. Furthermore, performance metrics such as the mixed processing time, the execution time and the percentage of idleness have been formulated in order to evaluate the system's behaviour.

The remainder of this paper is organized as follows: In Section 2 the simulation framework is described. The basic components are reported along with the state variables of the system. The principles of modelling the central processing unit (CPU) are also defined. Section 3 details the used metrics, outlines the performed

Manuscript received April 5, 2007
Manuscript revised April 25, 2007

experiments and discusses their results, whereas, Section 4 summarises our work. Finally, Appendices I and II respectively present the essential classes of the implemented framework and briefly describe the XML configuration file, in order to make our work easily understood.

2. Simulation framework

The study of the examined load-balancing algorithms is realized by the development of a proper simulation framework based on the event generation in certain time points (discrete event simulation) [13], [14], [15]. It is implemented by using the Java programming language with the assistance of the core features of the J-Sim platform [16], and supports the simulation of simple load balancing algorithms with the following characteristics:
i) They follow centralized policies [17]. A central entity is responsible for collecting the information about the state of the system and choosing the most appropriate candidate for the task assignment.
ii) They do not apply any special selection policy; the tasks are generated and sequentially dispatched to the proper processor.
iii) The update of the system's state information takes place either by explicit requests of the entity in which this information resides (Demand-driven policy), or by the processors in case their states change by a certain degree (State-change-driven policy) [17], [8].
iv) The location policy is performed via a function, the efficiency function, which takes into account the features and the state of the processor (see Appendix I).

A presentation of the fundamental classes of our framework is laid out in Appendix I. Furthermore, Appendix II presents a short description of the XML configuration file by which the parameters of the simulated system are defined.

2.1 Basic Components

In order to simulate a computing system, the implementation of at least two vital entities is required; the processor (or computing unit) and the task. Table 1 outlines the quantities which describe the state of these entities, comprising the fundamental state variables [15] of the system, along with those which designate their features. The latter arise mainly from the model of the central processing unit (CPU) as it is presented in Section 2.2.

Table 1 - Stare variables *(s)* and features *(f)* of the simulated entities

Processor	*(f)***Resource units**: The magnitude of the resource of the processor.
	*(f)***Processing speed**: the processing speed, in arbitrary units.
	*(f)***Topology factor**: the "distance" from a reference point (in current model the reference point is the entity that is responsible for the task generation), in arbitrary units. This magnitude results from the requirement of a general expression for the delay originating from the communication among the components of the system.
	*(f)***Delay per topology factor unit**: the delay, in time units, per topology factor unit. It is realized in order to quantify the topology factor with specific metric units.
	*(s)***Remaining task units**: the task units that remain to be executed by the processor.
	*(s)***Active tasks**: the number of the tasks that are still being processed.
	*(s)***Completed tasks**: the number of the tasks that has already been processed.
	*(s)***Rejected tasks**: the number of tasks which is rejected from the processor due to their non-compliance with its policy.
Task	*(f)***Task units**: the magnitude of a task, in arbitrary units.
	*(f)***Priority**: the execution priority.
	*(s)***Generation time**: the time of the generation of the task.
	*(s)***Arrival time**: the time when the task arrives at the processor.
	*(s)***Completion time**: the time when the processor completes the execution of the task.

2.2 The CPU model

We assumed that the simulation model of the CPU of the processor complies with the Proportional Share Resource Allocation policy (a time-shared policy) [18], which is based on the following statements:
i) The resource is split into discrete quantities which are bound, for certain time units, to process a task.
ii) For each task, the weight defines the relative percentage of the resource which is allocated for this task per time unit.
Thus, if w_i represents the weight of the task i, and $A(t)$ the set of the active tasks at the time t, then $f_i(t)$ denotes the fraction of the resource that is committed for processing the task i at the time t; as it is expressed by Equation 1.

$$f_i(t) = \frac{w_i}{\sum_{j \in A(t)} w_j} \quad (1)$$

In case $f_i(t)$ remains constant during the time period $[t_1, t_2]$, the task i could use the resource for $(t_2 - t_1) \times f_i(t)$ time units. The implementation of this model evinces the need to define quantities which depict the resource of the processor, the weight of the task, and its size. Moreover, an association between the size of the task and the required percentage of the resource for its completion should be specified. Therefore, the following definitions are made:

i) The resource of the processor consists of a set of units, the **resource units**.

ii) The magnitude of a task is represented by a set of units, the **task units**.

iii) A task allocates the maximum number of available resource units.

iv) One time unit is the time period which is required by one resource unit to process one task unit. The aforementioned relation is valid for the processor, the so-called baseline processor, which has the processing speed of one unit.

v) The **processing speed** is denoted by a positive integer number which indicates the speed ratio between a processor and the baseline.

vi) The weight of a task is represented by the **priority**.

According to the previous analysis and during a random period of time Δt_m in which the number of active tasks in the processor remains constant, the following equations are derived. Equation 2 issues the available resource units cpu_i for the execution of the task i. It is evaluated by the ratio of the task's priority p_i over the sum of the priorities of all the active tasks n, multiplied by the resource units cpu of the processor. Whereas, Equation 3 calculates the task units of the task i, tu_i^m, which are processed during Δt_m. The quantity tu_i^m is proportional to Δt_m, the processing speed sp of the processor, and the allocated resource units for the task i.

$$cpu_i = \frac{p_i}{\sum_{j=1}^{n} p_j} \times cpu \quad (2)$$

$$tu_i^m = \Delta t_m \times cpu_i \times sp \quad (3)$$

The elapsed time between the arrival and the completion of the task i at the processor (the mixed processing time, see Section 3.1) is given by the $\sum_{m=k}^{u} \Delta t_m$, where $\sum_{m=k}^{u} tu_i^m = tu_i$ and tu_i the size of the task. In the study by Gomoluch et al. [19] the CPU is modelled likewise.

3. Experiments

This section presents the results from the simulation of two load balancing algorithms. The extensive study of the influence of parameters, such as communication speed and topology, on the system's efficiency, discloses valuable conclusions which could be generalized to any algorithm which uses similar mechanisms. The examined algorithms are described as follows:

i) The deterministic Round-Robin algorithm (RR) [7]: RR does not take into account the state of the system since the first task is assigned to the first processor and so on. The exchanged messages are limited to those which are required for the task dispatching (one message per task).

ii) The LUIA: Our algorithm could be characterized as centralized [17] since the location and the information policies are performed by a single entity. It does not abide by any particular selection policy; the information policy is sequentially applied to the tasks according to their generation time, and their processing starts immediately on their arrivals at the computing units. The most suitable candidate for the task assignment becomes the one which is able to complete its remaining task units in the shortest period of time. This criterion is expressed by the efficiency function $f_x = totalLoad / speed$, where *totalLoad* and *speed* are the remaining task units for processing and the processing speed of the processor respectively. The required information for the evaluation of *fx*, is acquired by explicit requests to the processors by the entity in which this information resides. Therefore, for each task, $2 \times n + 1$ messages are exchanged (n pairs of request and response, one of each processor, and one message for the task dispatching).

The task generation is realized by an entity, the generator, which does not participate in the processing of the tasks. This entity also applies both the information and location policy, justifying the titles "informant" and "regulator" respectively which are additionally used to designate its functionality in this paper. Furthermore, it is considered to be the reference point for the construction of the grid, so the locations of the remaining entities are relative to it. Figure 1 illustrates LUIA based on the software components of the simulation framework (see also Appendix I).

Note that, although a Computational Grid could be composed by thousand processors, in our simulations it consists of a few ones only for simplicity reasons, since the results for the selected topologies, could be generalized.

3.1 Performance metrics

The subsequent paragraphs present the quantities, the performance metrics, which we have set in order to evaluate system performance. Some conclusions deduced from their study are also included. It is assumed that the selection policy complies with the nonpreemptive transfer [20], [21].

The **mixed processing time** is the elapsed time between the arrival and the completion of the task at the processor. It is essentially dependent on the percentage of the resource units and the time period devoted to each task execution. For example, if the resource is divided equally into two tasks, their respective mixed processing times are twice as much compared with those of in the case these tasks are processed individually. During this time the processor is *busy*, as its resource units are bound to process the applied workload. The **processing time** of a task is the actual time that the processor allocates for its completion. It is exclusively dependent on the task size and the processor's features, since the relations among the speed, the resource and the task unit are predefined. The **execution time** is the elapsed time between the generation and the completion of the task. For the above metrics, the respective average metrics over the number of the completed tasks in a processor are defined as mean mixed processing, mean processing and mean execution time (processor-based metrics). The **percentage of idleness** (**%idle time)** is the percentage of the time period over the simulation time, in which the processor is free of tasks. Note that the simulation time is denoted as the elapsed time from the generation of the first task until the completion of all of them. Figure 2 illustrates the aforementioned metrics accompanied by some clarifications. These metrics could be generalized to the entire system (system-based metrics). For example the %idle time could be defined as the percentage of the time that the system is inactive over the simulation time. It derives from the intersection of the respective idle time periods of the processors.

The comparison of the mean processing and the mean mixed processing time designates whether the processor responds properly or not to the applied workload. A slight deviation between them signifies that the resource is distributed to a few tasks, indicating that the performance of the processor is probably satisfactory. Furthermore, the combined examination of the mean execution time and the mean mixed processing time denotes whether the negotiation protocol converges quickly or not. Finally, a great value of %idle time possibly indicates that the processor does not participate significantly in the process. This metric should be evaluated concerning the completed tasks since the idleness could be attributed to the high processing speed. In this case, the potential removal of the processor could induce the overloading of the system and consequently, performance degradation.

3.2 Communication speed influence

The communication speed among the entities has an essential role to the performance of the system, especially in cases where the negotiation protocol involves the exchange of great numbers of messages. This is verified by the comparison of simulation results of the aforementioned algorithms. The parameters of the simulated system are included in Table 2. The values of the *delay* and the task size vary from 0.01 to 0.1 sec and 85 to 160 task units (t.u.) with increase step 0.01 sec and 15 t.u. respectively.

Table 2 - Simulation parameters

<table>
<tr><th colspan="5">Grid</th></tr>
<tr><th>Number of Processors</th><th>Resource units (cpu)</th><th>Processing speed (sp)</th><th>Topology factor (tf)</th><th>Delay per topology factor unit (delay) (in sec)</th></tr>
<tr><td>3</td><td>100</td><td>1</td><td>1</td><td rowspan="2">0.01...0.1 step: 0.01</td></tr>
<tr><td>2</td><td>100</td><td>3</td><td>1</td></tr>
<tr><th colspan="5">Tasks</th></tr>
<tr><td colspan="2">Task generation distribution</td><td colspan="3">constant rate: 10 tasks/sec</td></tr>
<tr><td colspan="2">Task units distribution</td><td colspan="3">task units(t.u.): 85...160/step: 15</td></tr>
<tr><td colspan="2">Priority distribution</td><td colspan="3">constant priority: 1</td></tr>
<tr><td colspan="2">Task generation time</td><td colspan="3">300 sec</td></tr>
<tr><td colspan="5">Critical simulation parameters:
The selection of the task generation distribution along with the delay takes place in such a fashion that: in RR, the execution time of the smallest task in the processor with the lowest sp is greater than the task inter-arrival time to that processor.
Assuming that a task of 85 units is processed alone in the slowest computing unit, its execution time is $\frac{85}{100\times 1} = 0.85\text{sec}$.
The task inter-arrival time:
(total number of processors) × (task generation rate)$^{-1}$ = $5\times 0.1 = 0.5\text{sec}$. Thereby, the aforementioned condition is satisfied since 0.85 > 0.5. Otherwise, the system is capable of processing each task individually and the simulation results are without purpose.</td></tr>
</table>

The curves in Figure 3 illustrate the variation of the mean execution time with respect to *delay* per task size, for the applied algorithms. For example, LUIA(160) denotes the mean execution time, of the LUIA under the workload of 160 t.u. It should be mentioned that the curves of RR which illustrate considerably inferior system performance than the respective ones of LUIA (RR(115) to RR(160)), are excluded in favour of the figure's simplicity.

In LUIA, the performance of the system is profoundly affected by the pending time (*PQ*) of the tasks in the generator's queue until the application of the information policy. This time is inclusively determined by the quantity ((task generation rate)$^{-1}$) (*SG*) and the entailed time in information assembly (*TP)*. The relation between *SG* and *TP* controls the grid's behaviour, which is to be examined presently.

In our model, *TP* is twice the amount compared with the time that the informant needs to communicate with the most distant processor. This occurs since the termination of the processors' state retrieval is triggered upon receipt of the last status message (see Appendix I). *SG* is 0.1 sec, and all the processors have a topology factor equal to one unit, so the value of the *delay* where the equality *TP* = *SG* takes place is 0.05 sec. This time, called as behaviour threshold time, it is characteristic of this system, and it is denoted by t_b.

In the case that *SG*>*TP*, the generator's queue is empty. The information policy is applied directly to a newly produced task and the appropriate processor is selected in *TP* sec. In this scenario, the execution time essentially depends on the task generation rate, the workload and the features of the processor, whereas the communication speed has a secondary role. This behaviour is confirmed in Figure 3, where for *delay* < t_b, the mean execution times are virtually the same.

On the contrary, when SG<TP a task is dispatched to the processor within time greater than TP, since the information policy is not immediately applied to this task. In this situation, the efficiency of the system is controlled by PQ and the applied workload. Low values of PQ and high workload can improve it, as the delayed arrival of the tasks at the processors causes an increment of the available resource units per task. This provokes the diminution of the mixed processing time and subsequently the improvement of the execution time. The curves LUIA(130), LUIA(145) for 0.05<*delay*<0.07 and LUIA(160) for 0.05<*delay*<0.08, are considered characteristic examples. In the case of low workloads, the influence of PQ on the reduction of the mixed processing time is not significant, since the processors are already capable of processing them quickly enough. The performance is mainly controlled by PQ causing its degradation as LUIA(85) depicts. There are also some cases where a balance between the mixed processing time and PQ is achieved, so that the mean execution time will remain finally constant (LUIA(100), 0.05<delay<0.07). Nevertheless, high values of PQ, irrespective of workload, cause the deterioration of system efficiency since TP is the dominant stage of the whole procedure; an example comprises the whole series of the LUIA, for delay > 0.07 sec, where all the curves eventually assimilate into a single one. Furthermore, the following conclusions can be deducted.

In a static algorithm, like RR, the mean execution time is not affected by the communication speed, since there are no message exchanges among the processors.

In systems where the decisions are accomplished by exchanging large numbers of messages, the influence of communication speed on their performance is decisive. LUIA(145) shows that an increase of the *delay* by 0.01 sec could cause a change in the percentage of the average execution time, up to 32% approximately.

A dynamic algorithm may have considerably better performance than a static one apart form the cases where the communication overhead has a negative impact on the progress of the negotiation protocol. For example, LUIA outperforms RR, with the exception of some regions of the LUIA(85) where the high *delay* (*delay*>0.8) and the low workload give an advantage to RR. Eager et al. [12] have also concluded that extremely simple load sharing policies perform virtually as well as complex ones which utilize more system information.

Finally, it should be mentioned that the increased delay can enhance system efficiency under high workload, an observation which has also been verified in a previous study [9]. This becomes evident in Figure 3 where the curves LUIA(115) to LUIA(160) display that the minimum values of the average execution time reside in the region in which the delay has a dominant influence on the system's performance (*delay* > t_b).

3.3 Topology influence

Topology is regarded as one of the utmost important factors since it could strongly affect grid performance. A measure of its impact can be the convergence of the location policy resolutions (with respect to diverse topologies), provoked by the elimination of the out-of-date state information (or stale information) problem [17], [22]. This aspect is accordingly clarified in the subsequent sections.

In our experiments, the grid which is considered has two groups of processors at its disposal. The group of high and low speed, which in short are called VPCs(high) and VPCs(low) respectively. The difference between them is spotted in the processing speed. The processors of VPCs(high) have greater speed than those of VPCs(low). The examined topologies are two. The first one brings

VPCs(high) to a comparatively shorter distance from the regulator than VPCs(low), and is called **near layout** (NL), whereas the other, which is called **far layout** (FL), works the opposite way. For each topology, two experimental series are carried out applying the LUIA algorithm. The difference between them lies in the absence or presence of the out-of-date state information problem by accordingly introducing the proper correction factor in the efficiency function.

3.3.1 Correction factor evaluation

In LUIA the deviation between the real state of the system and the state where the decisions of the location policy are derived from, is attributed to the time difference between the record and the use of the required information for that decision making. In the ideal scenario where the regulator acknowledges the state of every single processor the moment that a task arrives there, the decisions are based on accurate information. This scenario is accomplished by calculating the divergence, *ΔStatus*, between the processor's state at the time when it receives the request for recording it, and that when the arrival of the task takes place there.

In our model, the estimation of *Δstatus* is realized by the evaluation of the difference of the remaining task units, since this is the only state variable that takes part in the efficiency function. For each processor, let VPC_x, $\Delta Status_x$ is formed cumulatively in the following three time periods. The quantities tf_x and *delay* denote the topology factor and the delay per topology factor unit of VPC_x respectively.
i) Response time ($t_{resp(x)}$). It represents the time period that VPC_x requires to send a status message to the informant. The evaluation of this quantity is performed by the product of tf_x, and *delay*, as it is expressed by Equation 4.

$$t_{resp(x)} = tf_x \times delay \tag{4}$$

ii) Pending time ($t_{pend(x)}$). It defines the time period that the VPC_x's status message suspends in the informant until the completion of information retrieval procedure. This procedure, the duration of which is expressed by the Equation 5, is terminated with the reception of the last status message by the informant, which inevitably comes from the most distant processor, VPC_{max}.

$$t_{assem} = 2 \times tf_{\max} \times delay \tag{5}$$

Equation 5 derives from Equation 6 which expresses the required time, $t_{assem(x)}$, for the informant to acquire the information of the VPC_x. It is represented by the sum of the time period, $t_{req(x)}$, which the informant's request for a status message requires to arrive at VPC_x, and the time period $t_{resp(x)}$ that the VPC_x's reply needs to get into the informant.

$$t_{assem(x)} = 2 \times tf_x \times delay \tag{6}$$

Finally, the pending time $t_{pend(x)}$ equals to the subtraction of the time period that informant needs to gather the information of the VPC_x from which it is required to form the system state on the whole. Thereby, the combination of Equations (5) and (6), results in Equation 7.

$$t_{pend(x)} = 2 \times delay \times (tf_{\max} - tf_x) \tag{7}$$

iii) Sending time ($t_{send(x)}$), which represents the requisite time period for a task to arrive from the regulator to the VPC_x, and it is expressed by Equation 8.

$$t_{send(x)} = tf_x \times delay \tag{8}$$

Thus, the task units that are completed from VPC_x during the time period $t_{resp(x)} + t_{pend(x)} + t_{send(x)}$, express $\Delta Status_x$ (Equation 9). This quantity represents the correction factor that should be taken into account for the evaluation of the remaining task units of the VPC_x.

$$\begin{aligned} \Delta Status_x &= (t_{resp(x)} + t_{pend(x)} + t_{send(x)}) \times cpu_x \times sp_x \\ &= 2 \times delay \times tf_{\max} \times cpu_x \times sp_x \end{aligned} \tag{9}$$

where cpu_x and sp_x, represent the resource units and the processing speed of the processor VPC_x respectively.

Assuming that t_1 is the time when a VPC_x receives the informant's request for recording and dispatching a message with its state to it, and $TUs(t_1)$ is the remaining task units of the VPC_x, the reported value TU_{Sr} is expressed by Equation 10.

$$TUs_r(t_1) = TUs(t_1) - \Delta Status_x \tag{10}$$

3.3.2 Decision differentiation

The research of the cases under which the result of the comparison between two processor units (VPC_1 and VPC_2) depends on the applying efficiency function, leads to the detection of the system's decisions convergence due to the insertion of the correction factor. In our experiments, the quantification of a VPC_i's efficiency, $fx_i(t)$, at the time point *t*, is performed by Equation 11.

$$fx_i(t) = \frac{TUs_i(t)}{sp_i} \tag{11}$$

where $TUs_i(t)$ and sp_i, signify the remaining workload and the processing speed of the processor VPC_i respectively. The evaluation of the above relation enacts virtually a metric of the time that VPC_i needs to accomplish TUs_i task units. The combination of Equations 9, 10 and 11 results in Equation 12 expressing the efficiency function, fxc_i, which encapsulates the correction factor.

$$\begin{aligned} fxc_i(t) &= \frac{TUs_i(t) - \Delta Status_i}{sp_i} \\ &= \frac{TUs_i(t) - 2 \times delay \times tf_{\max} \times cpu_i \times sp_i}{sp_i} = fx_i(t) - C_S \end{aligned} \tag{12}$$

Considering that all processors have identical resource units, then $C_S = 2 \times delay \times tf_{\max} \times cpu$.

To determine when the variation in location policy decisions occurs, let fxc_1, fx_1 and fxc_2, fx_2 be the efficiency functions with and without the correction factor, for the processors VPC_1 and VPC_2 respectively, and let us consider the following statements:

i) According to location policy, the most proper processor for the task assignment is considered to be that with smaller fx_i, whereas, in the case of equality the one with the higher processing speed is chosen.

ii) Zero value of the fx_i or *fxci* indicates a processor which is in idle state.

iii) The relation $fx_i < C_S$ ($fxc_i = 0$) affirms that VPC_i gets to the idle state at best for the remaining time (commencing from the time that VPC_i receives the request for sending a status message to the *informant*) which is necessary for the completion of the information and location policy.

iv) The inequality $fx_1 < fx_2$ denotes that the execution time of the remaining workload of VPC_1 is less than that of VPC_2 (VPC_1 comes to idle state faster than the VPC_2, when the relation $fxc_1 = fxc_2 = 0$ holds additionally).

The variation in location policy decisions occurs when $fx_1 < fx_2 < C_S$ ($fxc_1 = fxc_2 = 0$) and the compared processors differ in processing speed. Indeed, presuming that VPC_1 has lower processing speed than VPC_2, the use of fx_i entitles VPC_1 as the most convenient candidate (since $fx_1 < fx_2$), whereas the application of fxc_i inducts VPC_2 (since the processors are idle and $sp_2 > sp_1$, where sp_1 and sp_2 are the processing speeds of VPC_1 and VPC_2 respectively).

In conclusion, when the correction factor is present, the system tends to allocate more tasks to the high speed processors, as compared to those allocated during its absence.

3.3.3 Cost Evaluation

It was proved that the differentiation of the decisions, with the use of *fxc* instead of *fx*, has as a direct outcome the assignment of additional tasks to VPCs(high). The following paragraphs demonstrate how this divergent behaviour, namely the case where the location policy delegates the task to an idle low speed processor (VPC_1) instead of an idle high speed processor (VPC_2), affects the system's performance concerning the topology. These alternative behaviours are illustrated in Figure 4.

The influence of the decision differentiation, in the way of cost or benefit to the grid's performance, could be represented by Equation 13.

$$cost(type) = topology(type) + execution(type) \quad (13)$$

where *type* defines the corresponding topology, near and far, for NL and FL respectively. The quantity *cost(type)* is the total cost/benefit, and *topology(type)* is the cost/benefit due to the topology. The latter is expressed by the task units that could have been processed or are processed by the processor as a consequent effect of its location in the grid. The term *execution(type)* designates the execution cost that emerges from processing a task in the low speed processor. It is assumed that in cases where the above quantities express benefit, their values are negative. Equation 13, the so-called cost layout function, serves as an indicator of the system's behaviour and is used to interpret the experimental results.

Concerning the NL, Equation 13 converts into 14

$$\begin{aligned} cost(near) &= cpu_2 \times sp_2 \times delay \times (tf_1 - tf_2) \\ &+ execution(near) \end{aligned} \quad (14)$$

where $topology(near) = cpu_2 \times sp_2 \times delay \times (tf_1 - tf_2)$ and $tf_1 > tf_2$. The term *topology(near)* depicts the task units that could have been processed by VPC_2 during its transfer to VPC_1, and consequently represents cost. The term *execution(near)* also expresses cost since the task is executed in the low speed processor.

Similarly, Equation 15 expresses the *cost layout function* for the FL.

$$\begin{aligned} cost(far) &= -cpu_1 \times sp_1 \times delay \times (tf_2 - tf_1) \\ &+ execution(far) \end{aligned} \quad (15)$$

where $topology(far) = -cpu_1 \times sp_1 \times delay \times (tf_2 - tf_1)$ and $tf_1 < tf_2$. An imminent task dispatching to VPC_2 would have as a consequence the deprivation of processing time equal to $delay \times (tf_2 - tf_1)$ time units in VPC_1. Therefore, *topology(far)* represents benefit. The term *execution(far)* reflects cost as long as the task is executed in VPC_1. In FL the system behaviour is diverse. In the case that $|topology(far)| > execution(far)$, the quantity *cost(far)* is negative. Consequently, the task is better executed in VPC_1, since the profit that could be obtained from the task completion in VPC_2 is scattered away by the time loss due to its transfer.

The above study demonstrates that topology is one of the uppermost factors that should be taken into account by a location policy. For instance, in cases where the decisions are made among idle processors with different processing speeds, the efficiency of LUIA algorithm (which does not consider the topology) would be improved if it took also into account the outcome of equations which are akin to Equations 14 and 15 (and does not simply choose the one

with the higher processing speed). On the contrary, when the processors have identical speeds, the system performance would be improved if the processor with the smaller topology factor was chosen.

3.3.4 Results

In this section, the aforementioned considerations are confirmed by the simulation of LUIA in a grid which its parameters are depicted in Table 3.

Table 3 - Simulation parameters

Grid				
Number of Processors	**Resource units** *(cpu)*	**Processing speed** *(sp)*	**Topology factor** *(tf)*	**Delay per topology factor unit** *(delay)* (in sec)
3	100	1	1/0	0.01...0.115 step: 0.005
3	100	2	0/1	
Tasks				
Task generation distribution	constant rate: 10 tasks/sec			
Task units distribution	task units (t.u.): 70			
Priority distribution	constant priority 1			
Task generation time	300 sec			

Critical simulation parameters:
The decision differentiation ($fx_1<fx_2<C_S$.) is reinforced by the proper adjustment of the workload. It is regulated so that the task assignment in VPCs(low) may take place marginally. This occurs when the execution time (t_{ex}) of a task in a high speed processor, is approximately equal to the sum of the time which is required for its potential re-election for a task assignment ($t_{reassign}$), and the necessary time (t_{send}) for transferring a task from the generator to the processor. Since VPCs(low) in this borderline situation is not included in the possible choices of location policy, the computation of the re-election time is based on VPCs(high). According to the former considerations, it is deduced that:

$$t_{reassign} = (task\ generation\ \ rate)^{-1} \times (number\ \ of\ \ high\ \ speed\ \ processors) = 0.1 \times 3 = 0.3\,\text{sec}$$

and

$$t_{send} = tf \times delay = (0\ or\ 1) \times 0.05 = 0\ or\ 0.05$$

The execution time of 70 task units equals to $t_{ex} = \frac{70}{cpu \times sp} = \frac{70}{100 \times 2} = 0.35\,\text{sec}$. Thereby, $t_{reassign} + t_{send} \cong t_{ex}$ indicating that a task of 70 task units is appropriate for performing the experiments.

The grid consists of two groups of processors, VPCs(low) and VPCs(high), which interchange in order to form the desired topologies, the NL and FL. For each topology a series of experiments is carried out, where the value of the *delay* ranges from 0.01 to 0.115 sec, with increase step 0.005 sec. The results of the simulation are depicted in Figures 5, 6, 7 and 8 where the following naming conventions are adopted. The terms fx, and fxc refer to experiments which were performed with the application of *fx* and *fxc*. The terms near, and far denote the topology of the grid, NL and FL, while low, and high are used to classify the group of low and high speed processors respectively.

Figure 5 illustrates the variation of the mean execution time over the *delay*, for both topologies and the applied efficiency functions. The *delay* is confined to the range [0.01, 0.05) since for values grater than 0.05, the execution times are essentially determined by the pending time of the tasks in generator's queue. Hence, the effect of the correction factor on the grid's efficiency is not distinctive. According to this figure, the following observations are made.

The divergence of curves fx-near and fxc-near increases with the augmentation of the *delay*. This behaviour complies with the description of Equation 14 since in NL, the choice of an idle low speed instead of high speed processor, always leads to a proportional reduction in the system's performance with respect to delay.

In FL, the examination of curves fx-far and fxc-far discloses that for $delay<0.035$ the mean execution times of the tasks which result from the application of *fxc* are better than the ones which derive from the *fx* usage. For the specific range of delay, the absolute value of the *topology(far)* factor is smaller than *execution(far)* indicating that the decision differentiation between *fx* and *fxc* has negative impact on the system's performance. The reduction of the divergence between the aforesaid curves while delay is enlarging is interpreted by the decrement of the difference between the quantities *topology(far)* and *execution(far)*, since *topology(far)* proportionally increases with delay, as it is expressed by Equation 15. For the marginal value of *delay*, 0.035 sec, where $topology(far) \cong execution(far)$, the system's performance is independent of the applied efficiency function, and the respective mean execution times are virtually the same.

The deviation between the curves fx-near and fxc-near is greater than the respective one of fx-far and fxc-far. This is an anticipatory behaviour, as in NL the decision differentiation has stronger impact on the system efficiency than in FL ($cost(near) > cost(far)$).

Figures 6 and 7 depict the variation of the average number of completed tasks and the average percentage of idleness

per processor group, over the *delay*, for the applied topologies and efficiency functions. The examination of these figures leads to the following conclusions.

The insertion of the correction factor in fx induces the diminution of the completed tasks; and consequently an increase in the percentage of the idleness of VPCs(low), regardless of topology, takes place. Considering that the total number of completed tasks remains constant, the above change provokes a reverse one of the respective quantities of VPCs(high). This is a presumed behaviour since *fxc* forces the system to assign more tasks to VPCs(high) than *fx* does.

The deviations of the curves fx-near-low, fxc-near-low and fx-near-high, fxc-near-high, for the variations of the completed tasks and the percentage of the idleness, are greater than the respective ones of the FL. This grounds as the influence of the decision differentiation on system's behaviour is greater in NL.

The out-of-date state information also plays a significant role in the configuration of the grid. Concerning Figure 7, for *delay*=0.075, the average values of the percentage idleness indicated by the curves fx-near-low and fxc-near-low are 62% and 96% respectively. Computing units with a high percentage of idleness are feasibly withdrawn from the system without any considerable change in its performance. Thereby, the application of *fxc* gives the choice of removing or using those processors in a different activity.

Figure 8 describes the variation of the mean execution time per processor group versus the *delay*, for *fx* and *fxc*, in NL. It is noted that the aforementioned variation is limited to the *delay*<0.05, whereas for higher values, the contribution of the decision differentiation to the mean execution time is negligible. This figure reveals that the diminution of the mean execution time with the insertion of the correction factor is essentially provoked by the decrease in the mean execution time of the VPCs(low), since the deviation of the curves fx-near-low and fxc-near-low is considerably greater than the respective one of fx-near-high and fxc-near-high. Considering that *fxc* provokes a kind of task transfers from VPCs(low) to VPCs(high), it is deduced that VPCs(low) appears to be more susceptible to the changes of the workload. It is clear that the cost of an imminent increase of tasks in VPCs(high) is less than the corresponding benefit which is achieved by their disposal of VPCs(low).

4. Summary-Conclusions

In the current work, a flexible simulation framework for the examination of certain types of load balancing algorithms is implemented and performance metrics are formulated. The interpretation of the experimental results of two algorithms, a classic static (RR) and a dynamic (LUIA) reveal the following points.

The communication delay and the workload can be regarded as two of the utmost issues that affect system performance, since they regulate and dictate directly or indirectly its behaviour. An algorithm should accordingly adjust its policies preserving high efficiency. For example, an algorithm resulting from the combination of LUIA and RR could operate better than either one individually.

The communication overhead emerging from the application of complex policies with the aim of ensuring up-to-date information, can eventually lead to adverse effects. A simple static algorithm, like RR, could be proved more effective than a dynamic one. Therefore, there is a compelling need to balance the cost of the acquisition and the accuracy of the information on which the decisions of an algorithm are based. Nevertheless, it appears that increased delay could benefit the system's efficiency under high workload.

Equations 14 and 15 confirm the interdependence between the topology and the performance of the system. It is shown that an algorithm should consider not only the state and the features of the processors, but also their relative positions in the grid.

The out-of-date state information problem should be appropriately dealt with since it provokes deficiency in the utilization of the system. For example, processors that could have been idle are instead occupied, depriving the capability of their use in a different activity.

Under the particular conditions of NL, it has been proven that disposing workload from processors with low processing speed, and allocating it to those with comparatively higher processing speed improves the system's performance.

Appendix I

In the following paragraphs, some of the most important classes are described in order for an overview of the system architecture to be formed.

VPC represents the processor (VirtualPC) and implements the CPU model. It reacts to three types of messages: one that represents a task (VTask), another being the message that holds the state of the processor (StatusMessage), and the third internally used for indicating the completion of a task. VPC uses StatusMessage to inform InfoCenter about its state. StatusMessage is dispatched by an explicit request from InfoCenter or by the percent variation of the processor's state variables over or below a specific threshold. Both the state variables and the threshold could be accordingly defined in the XML configuration file.

InfoCenter represents the entity which is dedicated to preserving a list with the states (state list) of the processors so that TaskGenerator can make the selection of the most appropriate one for the task assignment. It has two operational modes designating the two information policies (defined in the XML configuration file):
a) the LIVE_MODE where the decisions are based on the state of the system at the time when the request for the grid's information retrieval takes place. In this case, InfoCenter dispatches requests for receiving the status messages of all the processors. The evaluation of the best is performed when all the responses are received
b) the NO_LIVE_MODE where the decisions are realized according to the information that has already been stored in the state list.

TaskGenerator is responsible for the task generation and the task assignment. The task generation distribution is customized via the XML configuration file.

WMessage is the base class of the message hierarchy. It encapsulates the important events occuring during its life cycle (creation, destruction, departure and arrival at the various components), providing valuable information for the simulation progress.

VTask represents the task. Its life cycle consists of the following stages in order:
a) the generation stage, in which the task generation takes place
b) the processing stage, where the task has partially been completed
c) the completion stage, where the task has been completed.

StatusMessage realizes the message (status message), which contains the vital information (such as the active tasks, remaining task units for processing) about the state of a processor. The state list resides in the InfoCenter and actually consists of these kinds of messages. StatusMessage implements the method (efficiency function), which dictates the selection of the most appropriate candidate for the task assignment, ordering the status messages in the state list. The efficiency function is currently expressed by Equation 16.

$$fx = \frac{(tLCoef \times totalLoad)/(ntCoef \times nTasks)}{spCoef \times speed} + topoCoef \times topoFactor \quad (16)$$

where *totalLoad*, *nTasks*, *speed* and *topoFactor* are the remaining task units for processing, the number of active tasks, the processing speed and the topology factor respectively. The coefficients *spCoef*, *topoCoef*, *tLCoef* and *ntCoef*, and the implementation method are declared in the XML configuration file.
The application of the efficiency function comprises the default location policy. The user can externally either modify it or utilize his own policies by appropriately defining them in the processor selectors section (see Appendix II).

Appendix II

The simulation is dictated by an XML file (an example is depicted in Figure 9). This file consists of four sections, each of which is responsible for a particular configuration. A short description of these sections follows:

Processors section. This area describes the processors. The user could easily modify the features of a processor by assigning the desirable value in the appropriate attribute of the XML element. For example, the processing speed, the resource units, and the topology factor are housed in the "speed", "cpu_power", "topology_factor" attributes respectively. In addition, there are attributes which contain information about the processor's behaviour and the applied policies. For example, a positive value of the attribute "above_notify_thres" defines the threshold of the percent variation of the state variable (the remaining workload or the number of active tasks), above which the processor sends a status message to InfoCenter, whereas a negative value deactivates the aforementioned procedure.

Task inter-arrival section. The methods used for the estimation of the task generation distribution are described in this section. The user should accordingly define the implementation class and the method name as values of the attributes "class_name" and "method_name" respectively of the element "arrival_gen_functions". Moreover, there is the capability of declaring an array of parameters, in the form of name/values pairs, which could be invoked and utilized inside the method.

Processor selectors section. The methods used for the selection of the proper processor for the task assignment

are defined here. Its descriptive rules are similar to task inter arrival section.

Tasks section. The features of the generated tasks (task units and priority) are described in this area either directly by assigning the value, or indirectly by indicating the implementation class and the method which returns its value.

References

[1] Frank C. H. LIN and Robert M. Keller, "The Gradient Model Load Balancing Method", IEEE transactions on software engineering vol. SE-13, No. 1, January 1987.

[2] Hans-Ulrich Heiss and Michael Schmitz, "Decentralized Dynamic Load Balancing: The Particles Approach", Information Sciences vol. 84, issue 1&2, 1995, pp. 115-128.

[3] Phillip Krueger and Niranjan G. Shivaratri, "Adaptive Location Policies for Global Scheduling", IEEE Transactions on Software Engineering, vol. 20, no. 6, June 1994, pp. 432-444.

[4] Chin LU and Sau-Ming LAU, "An Adaptive Load Balancing Algorithm for Heterogeneous Distributed Systems with Multiple Task Classes", IEEE Proceedings of the 16th International Conference on Distributed Computing Systems, 1996, pp. 629 –636.

[5] Hwa-Chun Lin and C.S. Raghavendra, "A Dynamic Load-Balancing Policy With a Central Job Dispatcher (LBC)", IEEE Transactions on Software Engineering, vol. 18, no. 2, February 1992, pp. 148-158.

[6] Gurdeep S. Hura, Sheeja Mohan and T. Srikanthan, "On Load Sharing in Distributed Systems: A Novel Approach", Transactions of the SDPS, March 2002, Vol. 6, No. 1, pp. 59-81.

[7] Philip J. Rasch, "A Queuing Theory Study of Round-Robin Scheduling of Time-Shared Computer Systems", Journal of the ACM, vol. 17, no.1 , pp. 131-145, January 1970.

[8] Orly Kremien and Jeff Kramer, "Methodical Analysis of Adaptive Load Sharing Algorithms", IEEE Transactions on Parallel and Distributed Systems, Vol. 3 , No. 6, 1992, pp 747-760.

[9] R. Mirchandaney, D., Towsley and J. A. Stankovic, "Analysis of the effect of delays on load sharing", IEEE Transactions on Computers, vol. C-38, no. 11, November 1989, pp. 1513-1525.

[10] Sayed A. Banawan and Nidal M. Zeidat, "A Comparative Study of Load Sharing in Heterogeneous Multicomputer Systems", IEEE Conference Proceedings of the 25th Annual Simulation Symposium, April 1992, pp. 22-31.

[11] Songnian Zhou, "A Trace-Driven Simulation Study of Dynamic Load Balancing", IEEE Transactions on Software Engineering, vol. 14, no.9, 1988, pp. 1327-1341.

[12] D. L. Eager, E. D. Lazowska and J. Zahorjan, "Adaptive Load Sharing in Homogeneous Distributed Systems", IEEE Transactions on Software Engineering, vol. SE-12, no. 5, May 1986, pp. 662-675.

[13] Thomas J. Schriber and Daniel T. Brunner, "Inside Discrete-Event Simulation Software: How it Works and Why it Matters", Proceedings of the 2005 Winter Simulation Conference, pp.167-177.

[14] Harry Perros, "Computer Simulation Techniques: The definitive introduction!", Computer Science Department NC State University, Raleigh, 2003, pp. 1-19.

[15] Richard M. Fujimoto, "Parallel and Distributed Simulation Systems", Wiley-Interscience, United States of America, 2000, ISBN: 0-471-18383-0, pp. 27-48.

[16] Hung-ying Tyan, "Design, Realization And Evaluation Of A Component-Based Compositional Software Architecture For Network Simulation", Phd thesis, Ohio State University, USA, URL http://www.j-sim.org/, 2002.

[17] Niranjan G. Shivaratri, Phillip Krueger, and Mukesh Singhal, "Load Distributing for Locally Distributed Systems", IEEE Computer, December 1992, pp. 33-44.

[18] Kevin Jeffay, F. Donelson Smith, Arun Moorthy, James Anderson, "Proportional Share Scheduling of Operating System Services for Real-Time Applications", in Proceedings of the 19th IEEE Real-Time System Symposium, Madrid, Spain, December 1998, pp. 480-491.

[19] Jacek Gomoluch and Michael Schroeder, "Market-based Resource Allocation for Grid Computing: A Model and Simualtion", 1st International Workshop on Middleware for Grid Computing (MGC2003), Rio de Janeiro, Brazil, June 2003.

[20] Phillip Krueger and Miron Livny, "A Comparison of Preemptive and Non-Preemptive Load Distributing", in 8th International Conference on Distributed Computing Systems, June 1988, pp. 123-130.

[21] M. Harchol-Balter and A. B. Downey, "Exploiting Process Lifetime Distributions for Dynamic Load Balancing", ACM Transactions on Computer Systems, 1996.

[22] M. Mitzenmacher, "How Useful Is Old Information", IEEE Transactions on Parallel and Distributed Systems, vol. 11, no. 1, Jan. 2000.

List of Figures

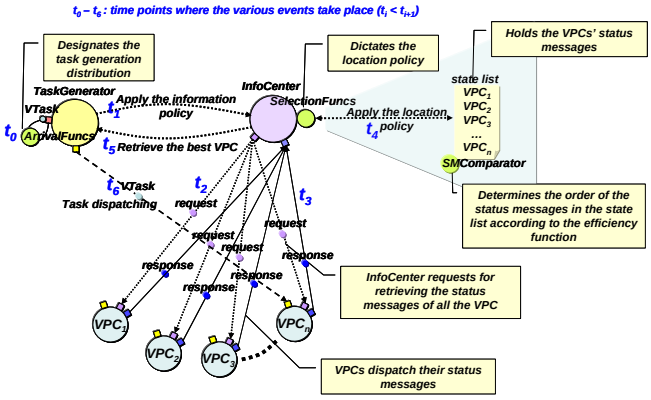
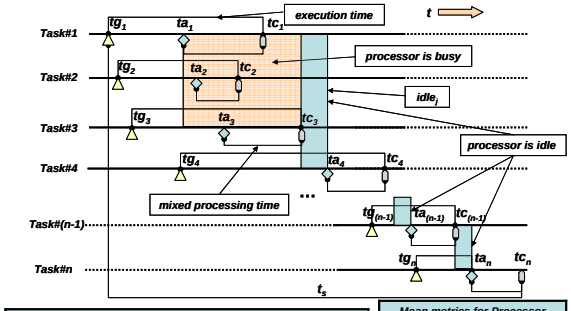


Figure 1 - The software component based behaviour of LUIA.



Fundamental times/ Metrics for a task i	
tg_i	The generation time
ta_i	The arrival time
tc_i	The completion time
$tm_i = tc_i - ta_i$	The mixed processing time
$te_i = tc_i - tg_i$	The execution time
$tp_i = tu_i/(cpu \times sp)$	The processing time of a task i with size tu_i in a processor with resource units cpu and processing speed sp

Mean metrics for Processor	
$\frac{\sum_{i=1}^{m} idle_i}{t_s} \times 100$	Percentage of idleness
$\frac{\sum_{i=1}^{n} tm_i}{n}$	Mean mixed processing time
$\frac{\sum_{i=1}^{n} te_i}{n}$	Mean execution time
t_s	Simulation time

Figure 2 - performance metrics.

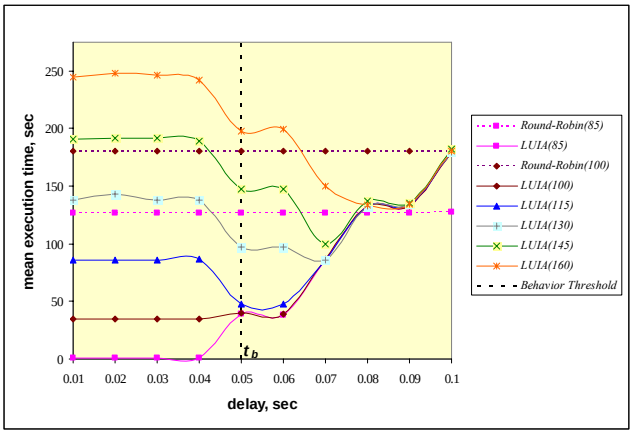


Figure 3 – Mean execution time versus *delay* and workload, for the applied algorithms.

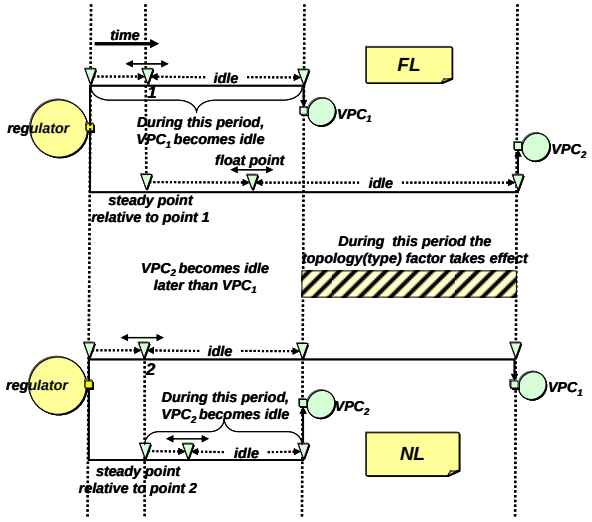


Figure 4 - The states of the processors during the decision differentiation.

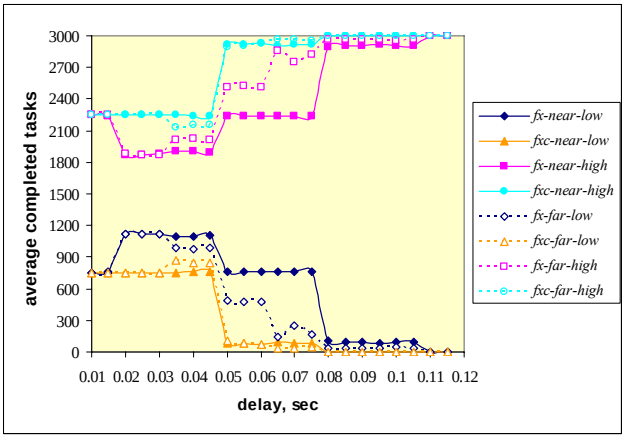


Figure 5 – Mean execution time versus d*elay.*

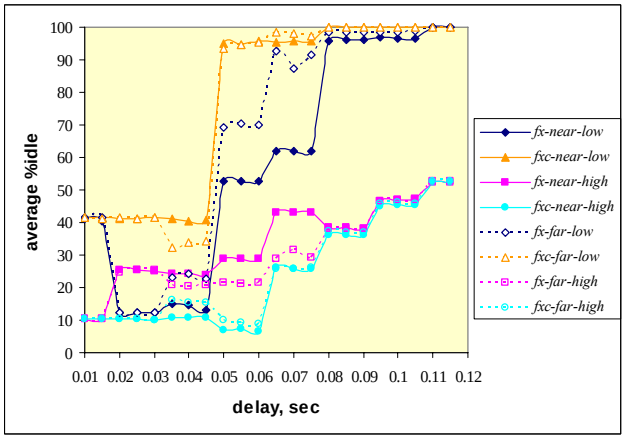


Figure 6 – Average number of completed tasks per processor group, versus *delay.*

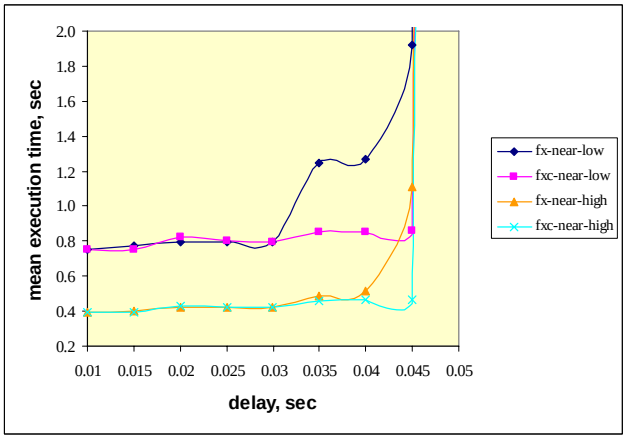


Figure 7 – Average of the percentage of idleness per processor group, versus *delay*.

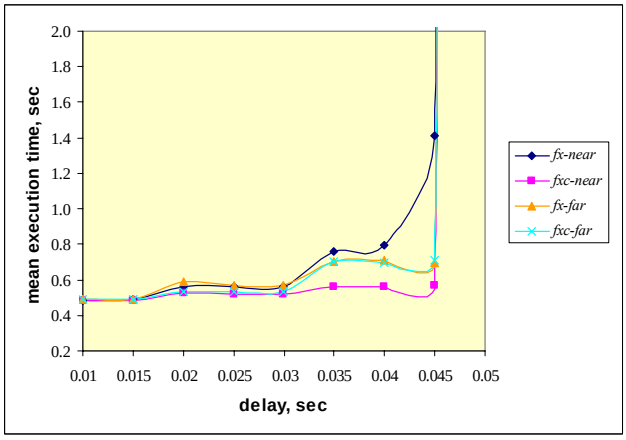


Figure 8 - Mean execution time per processor group, versus *delay*.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE configuration SYSTEM "config.dtd">
<configuration>
 <description>The configuration file for the creation of VPCs grid</description>

 <vpcs description="VPCS template">
  <vpc id="vpc1" replicate="2" speed="1" cpu_power="100" topology_factor="1"/>
  <vpc id="vpc2" replicate="3" speed="2" cpu_power="100" topology_factor="3"/>
  <rest_info cpu_thres="0." above_notify_thres="-0.1" mdelay_per_tf="0."/>
 </vpcs>

 <arrival_gen_functions description="function of tasks inter arrival">
  <fun_ar description="const_interval" class_name="vpcs.ArrivalFuncs" method_name="constIntervalFunc"
                                                                                    is_enable="true">
   <param description="interval" val="0.05"/>
   <task id="taskv1">
  </fun_ar>
 </arrival_gen_functions>

 <comp_sel_functions description="function of VPC selection">
  <fun_sel description="efficiency" class_name="vpcs.SelectionFuncs" method_name="effLiveUpdBasedSelection"
                                                                                    is_enable="true">
   <param description="speed_cf" val="1"/>
   <param description="topology_cf" val="0"/>
   <param description="tload_cf" val="-1"/>
   <param description="ntasks" val="0"/>
  </fun_sel>
 </comp_sel_functions>

 <tasks>
  <task id="taskv1" tunits="110" priority="1"/>
 </tasks>

</configuration>
```

Figure 9 – XML configuration file.

Ioannis D. Psoroulas was born in Athens, Greece in 1972. He received his Chemical Engineering Diploma from the National University of Athens (NTUA) in 1998. Since 2002 has been a PhD candidate in the Communications, Electronics & Information Systems department, division of Electrical and Computer Engineering of the NTUA. His research interests are in the fields of Simulation Systems, Grid Computing, Internet Technologies, and Multimedia Systems and Software.

Ioannis E. Anagnostopoulos was born in Athens, Greece in 1975. He received his Electrical and Computer Engineering Diploma from the University of Patras in 1998, and his Ph.D. from the School of Electrical and Computer Engineering of the National Technical University of Athens (NTUA) in 2004. In 2004, he joined the University of the Aegean as a visiting lecturer at the Department of Information and Communication Systems Engineering. His research interests are in the fields of Internet Technologies, Web Information Management, Communication Networks, and Multimedia Systems and Software.

Vassili Loumos received the Diploma of Electrical Engineering and the PhD degree in Computer Science from the National Technical University of Athens (NTUA), Greece. At present, he is a Professor at the NTUA, teaching Multimedia Technologies and Computer Graphics. His research and development activities focus on the fields of Image Processing and Computer Vision, Internet Technologies and Applications, and Computer Networks.

Eleftherios Kayafas received his B.Sc. degree from Athens University in 1970, and the M.Sc. and Ph.D. degrees in electrical engineering from the University of Salford, England, in 1975 and 1978, respectively. In 1979 he joined the National Technical University of Athens (NTUA) as Lecturer in the Electrical Engineering Department, then Asst. Professor (1987), Associate Professor (1992) and finally Professor of Applied Electronics (1996). His research interests are applied electronics, multimedia applications and multimedia communication systems. He has published more than 140 papers in journals and conferences, in the above subjects.