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

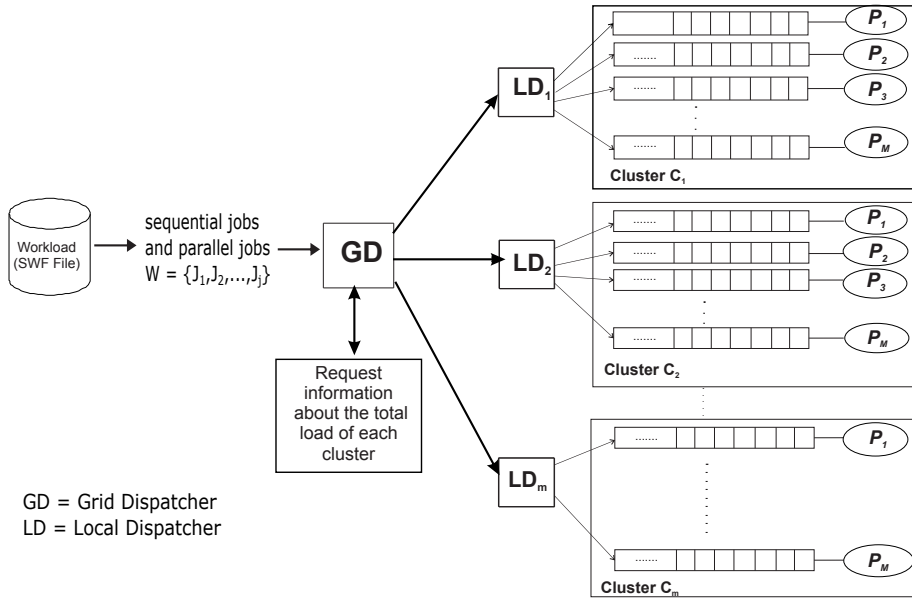

Figure 1. Multicore multicluster simulator model based on queues – GSMSim

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.

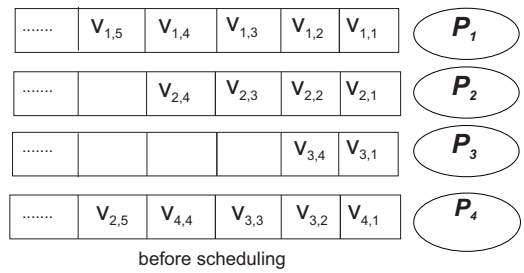

Figure 2. Jobs in queues – before scheduling

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 $O(n) + O(1) = O(n)$.

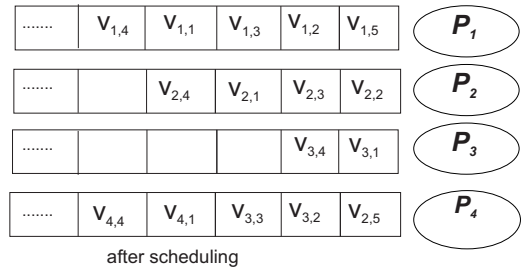

Figure 3. After scheduling – using AFCFS

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].

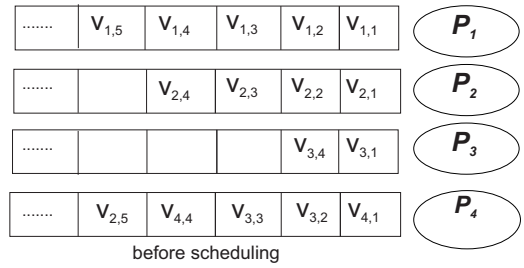

Figure 4. Jobs in queues – before scheduling

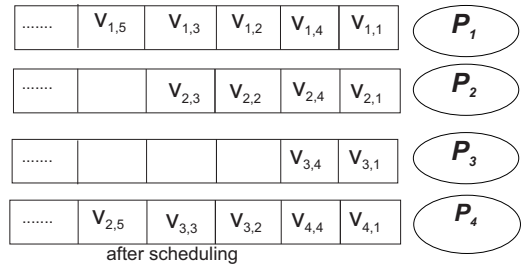

Figure 5. After scheduling – using LJFS

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.

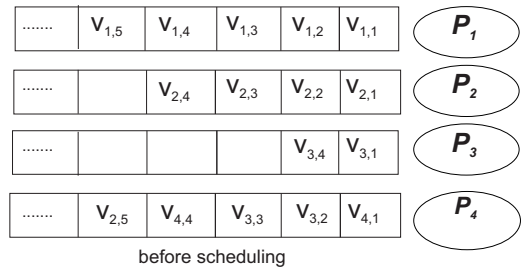

Figure 6. Jobs in queues – before scheduling

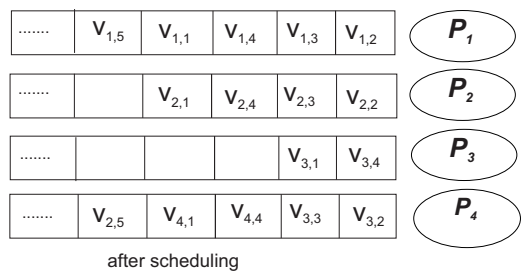

Figure 7. After scheduling – using LXF

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.

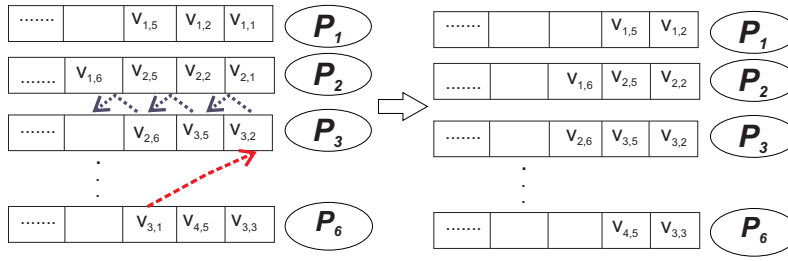

Figure 8. Example of a migration scenario

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.

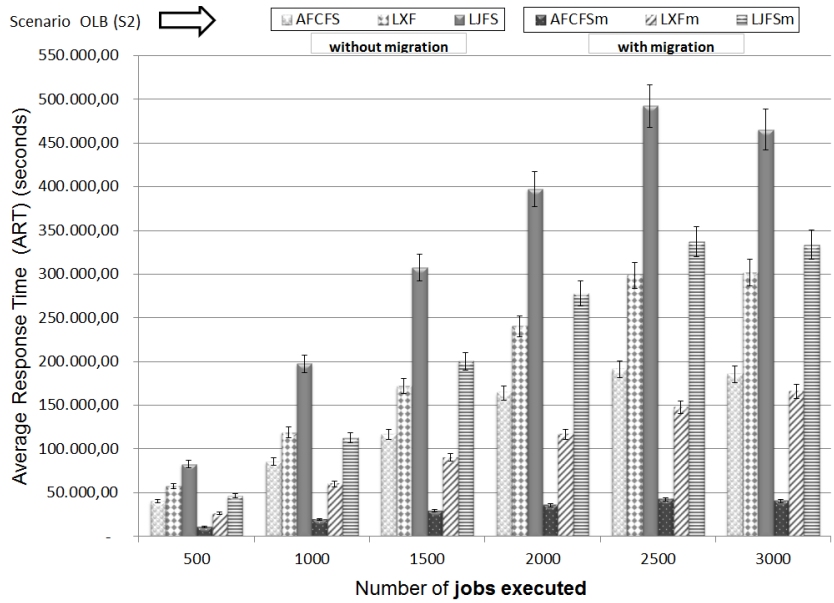

Figure 9. Scenario OLB – ART *versus* number of executed jobs

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.

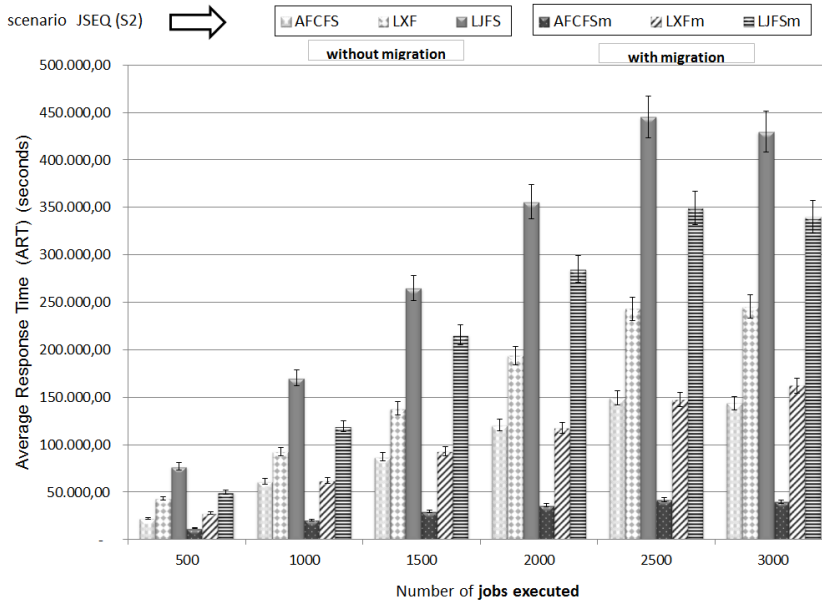

Figure 10. Scenario JSEQ – ART *versus* number of executed jobs

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.

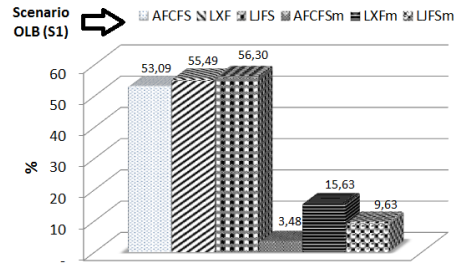

Figure 11. S1 – (%) LoC in the system

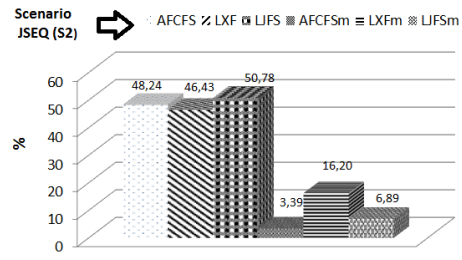

Figure 12. S2 – (%) LoC in the system

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.

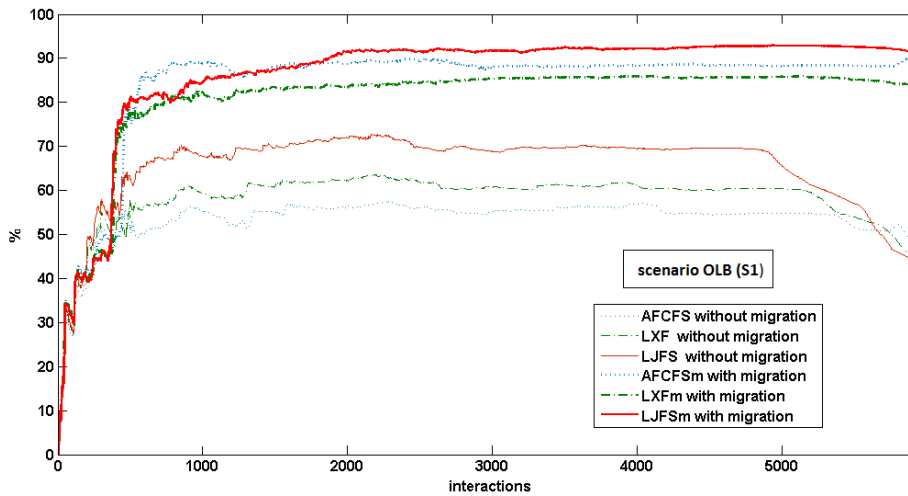

Figure 13. Scenario S1 – utilization rate (%)

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

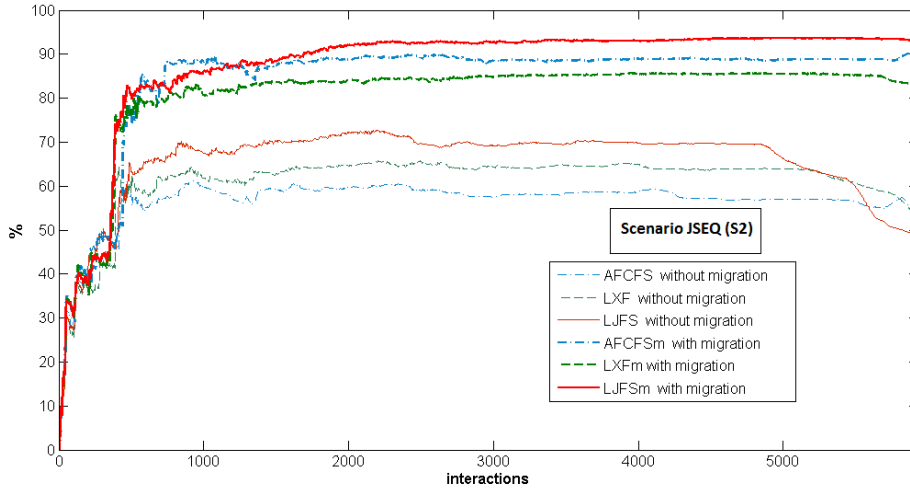

Figure 14. Scenario S2 – utilization rate (%)

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.