Reconfigurable Gang Scheduling Algorithm

Luís Fabrício Wanderley Góes[1], Carlos Augusto Paiva da Silva Martins[2]
Graduate Program in Electrical Engineering, Pontifical Catholic University of Minas Gerais
Av. Dom José Gaspar 500, Belo Horizonte, MG, Brazil
URL: www.pucminas.br
[1]{lfwg@uol.com.br} [2]{capsm@pucminas.br}

Abstract—

Using a single traditional gang scheduling algorithm cannot provide the best performance for all workloads and parallel architectures. A solution for this problem is the use of an algorithm that is capable of dynamically changing its form (configuration) into a more appropriate one, according to environment variations and user requirements. In this paper, we propose, implement and analyze the performance of a Reconfigurable Gang Scheduling Algorithm (RGSA) using simulation. The RGSA uses combinations of independent features that are often implemented in GSAs such as: packing and re-packing schemes (alternative scheduling etc.), multiprogramming levels etc. Ideally, the algorithm may assume infinite configurations and it reconfigures itself according to entry parameters such as: performance metrics (mean utilization, mean jobs response time etc.) and workload characteristics (mean jobs execution time, mean parallelism degree of jobs etc.). Also ideally, a reconfiguration causes the algorithm to output the best configuration for a particular situation considering the system's state at a given moment and based on past information. The main contributions of this paper are: the definition, proposal, implementation and performance analysis of RGSA.

Keywords— **Reconfigurable Algorithm, Gang Scheduling, Performance Analysis.**

1 INTRODUCTION

Nowadays, the service quality requirements of users and institutions increased. Thus, computer systems that provide many services (particularly, parallel machines) need to be highly utilized and provide a short response time for users jobs. Parallel job schedulers should match both requirements and workload (jobs) with resource availability (architecture, processors etc.) in order to maximize the system's performance. The main problem is that workload, requirements and resources change continuously. In order to solve this problem, many works have been developed to make job scheduling algorithms more flexible and adaptable [1], [9], [12], [13], [14], [17], [18], [20]. Up to now, a poorly explored solution is the use of reconfigurable computing concepts [3], [4], [13], [14], [16] in parallel job scheduling algorithms (like gang scheduling).

Reconfigurable computing emerged as a paradigm to fill in the gap between hardware and software, reaching better performance than software and more flexibility than hardware [3], [4], [16]. The reconfigurable devices including FPGAs (Field Programmable Gate Arrays) contain an array of computing elements or constructive blocks, whose functionalities are determined through the programming of configuration bits. Thus, an FPGA can implement different behaviors not established at design time. Because of this, reconfigurable devices (hardware) are improving the solutions for problems from different areas [3], [4], [16].

Our basic idea in this paper is to use reconfigurable computing concepts in a parallel job scheduling algorithm (gang scheduling) to maximize system's performance. According to a deep bibliographic revision [3], [4], [13], [14], [16], we found works that apply reconfigurable computing in software, but we did not find a previous work that used it on algorithms. In [13], we used a first approach to build a reconfigurable algorithm of a static parallel job scheduling algorithm. We improved this first approach to reach our present stage.

Ideally, the algorithm may assume infinite configurations and it reconfigures itself according to entry parameters such as: performance metrics (utilization, mean jobs response time etc.) and workload characteristics (mean jobs parallelism degree etc.). Also ideally, a reconfiguration causes the algorithm to output the best configuration for a particular situation considering the system's state at a given moment and based on past information.

Gang scheduling algorithms have been intensely studied in the last decade. They demonstrated many advantages over other parallel job scheduling algorithms, for instance, they: provide interactive response time for short jobs, through preemption; prevent long jobs from monopolizing processors; maximize the system's utilization etc [1], [2], [5], [6], [11], [12], [14], [18], [19], [20]. In our specific case it presents some interesting characteristics. It is composed of independent and well defined parts (packing and re-packing schemes, multiprogramming level, etc.) and each one has infinite possible solutions (implementations).

The **main objectives** of this paper are: to define, propose, develop and implement the RGSA; to analyze the performance of RGSA using simulation. The **main goal** is the implementation of RGSA in our simulation tool.

In this paper, we introduce the reconfigurable gang scheduling algorithm (RGSA) and relate it to other works in sections 2 and 3. In section 4, we present our experimental method: workload, metrics, configurations and parallel

architecture used in our simulations. Section 5 presents the experimental results and the performance analysis comparing RGSA and other traditional gang scheduling algorithms. Finally, in section 6 we highlight our conclusions and future works.

2 Reconfigurable Gang Scheduling Algorithm

Extending the reconfigurable hardware definition, we define a reconfigurable algorithm as an algorithm that is composed of constructive blocks, allowing its behavior to be modified through the form of its configuration.

A reconfigurable algorithm is composed of three layers: Configuration Control Layer (CCL), Reconfigurable Layer (RL) and Basic Layer (BL), as shown in Fig.1. The BL consists of a frame set and data structures. A data structure may be a list, a queue, an array or some structure that stores data. For example, in Fig. 2, a wait queue (data structure) stores jobs (data).

A frame represents a part or phase of an algorithm. For example, in a gang scheduling algorithm, a frame may represent a packing scheme that fits a job inside the Ousterhout matrix, which means it is only a part of a gang scheduling algorithm. There are two frame types: control and action frames. A control frame controls a specific characteristic of a data structure. In Fig. 2, the Multiprogramming Levels Frame controls the multiprogramming level of the Ousterhout Matrix. An action frame is responsible for process or move data between or inside data structures and frames. In Fig. 2, the Packing Schemes Frame receives a job from the Queue Policies Frame and fits it inside the Ousterhout Matrix.

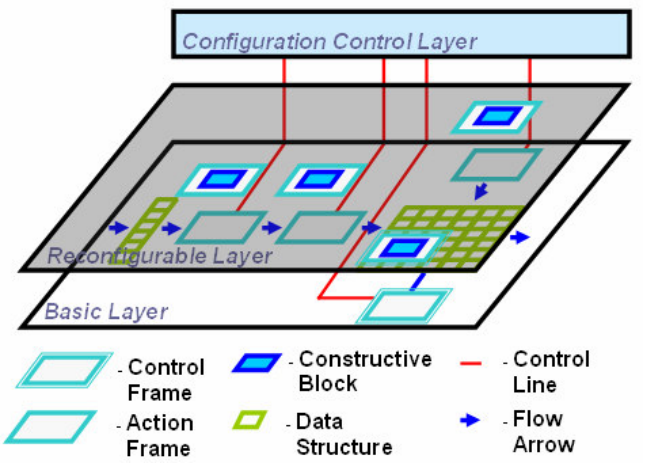


Fig. 1. The general architecture of a reconfigurable algorithm composed of three layers: Configuration Control Layer (CCL), Reconfigurable Layer (RL) and Basic Layer (BL).

The Reconfigurable Layer represents a configuration or an instance of the BL, in which every frame is filled out with one or more compatible constructive blocks at a certain moment. A constructive block is a possible implementation that can fill out with a specific frame. For example, the Re-Packing Schemes Frame, shown in Fig. 2, can be filled out with different re-packing schemes like slot unification and alternative scheduling, one at a time or simultaneously. So, each re-packing scheme implementation is a constructive block. When two or more constructive blocks simultaneously fill out a frame, they are executed in sequence. The maximum number of possible constructive blocks that fill out a frame is the number of different known implementations, for example, the number of known re-packing schemes.

The Configuration Control Layer chooses the constructive blocks that will fill out each frame at a given moment, thus it controls the configuration swapping. The choice is made based on entry parameters. The CCL can be implemented as a static table with pre-defined decisions, an evolutionary algorithm, a learning-based algorithm (neural network) etc. For example, we have a workload composed of long jobs and the most important metric for the user is the reaction time. So, the CCL will set a configuration that reduces the reaction time of the long jobs. In our case, the CCL should fill the Multiprogramming Levels Frame with the Unlimited Constructive Block, allowing a job to start its execution as soon as it was submitted.

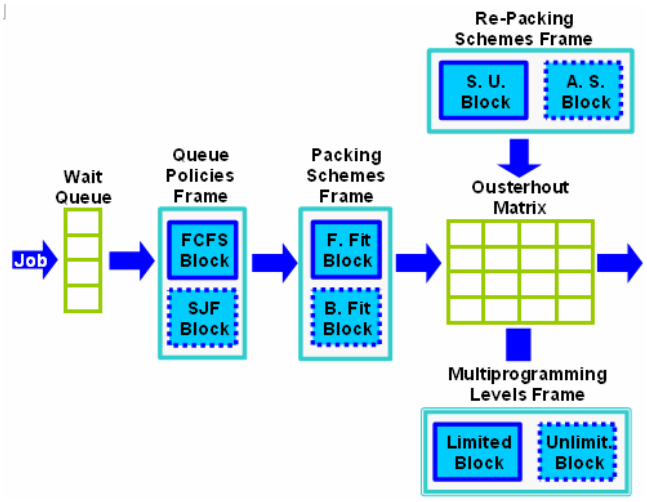


Fig. 2. The Basic Layer of the Reconfigurable Gang Scheduling Algorithm (RGSA) and some possible constructive blocks of the Reconfigurable Layer.

A gang scheduling algorithm may be composed of four parts: a packing scheme, a re-packing scheme, a queue policy and a multiprogramming level. In our Reconfigurable Gang Scheduling Algorithm (RGSA), as show in Fig. 2, each part is a different frame with two constructive blocks, to simplify our study. The first three are action frames and the last one is a control frame.

The Packing Schemes Frame may be filled out with two different packing schemes based on capacity: first fit or best fit. The Re-Packing Schemes Frame may be filled out with the slot unification and/or alternative scheduling re-packing schemes. The Queue Policies Frame can use the First Come First Served (FCFS) or Short Job First (SJF) policies. Finally, the Multiprogramming Levels Frame can be filled out with the Unlimited or Limited Multiprogramming Level Constructive Blocks.

In our RGSA, the CCL is implemented as a table (or switch case structure) that knows the best configuration according to some workload parameters, as shown in Table 1. The workload parameters and possible values are: execution time (high (H) or low (L)), parallelism degree (high (H) or low (L)), predominance degree (60%, 80% or 100%) and the most important metric (utilization (UT), reaction time (ReacT), slowdown (SD), response time (RespT) or simulation time (ST)). Then CCL evaluates these parameters and reconfigures RGSA to the best configuration. The workload parameters chosen and configurations will be better discussed in the Experimental Method section.

Table 1. The actual CCL implementation that chooses the best configuration according to some workload parameters.

Case	Workload Parameters				Configu-ration
	Metric	Execution Time	Parallelism Degree	Predominance Level	
1	UT or ST	High	Low	100	Conf 2
2	UT or ST	Low	High	80	Conf 2
3	RespT	High	High	80	Conf 2
4	UT or ST	Low	Low	100	Conf 4
5	RespT	Low	Low	80	Conf 4
6	UT or ST	Low	High	100	Conf 5
7	ReacT	High	High	100	Conf 5
8	ReacT or RespT or SD	High	Low	80	Conf 5
9	RespT	Low	High	60	Conf 5
10	ReacT or RespT	Low	High	80	Conf 5
11	ReacT	Low	Low	60 or 80 or 100	Conf 5
12	SD	Low	Low	100	Conf 5
13	SD or ReacT	High	High	60	Conf 6
14	SD or ReacT	High	Low	100	Conf 6
15	SD	Low	Low	60 or 80	Conf 6
16	SD	High	Low	60	Conf 6
17	RespT	High	High	100	Conf 7
18	RespT	High	Low	100	Conf 7
19	RespT	Low	Low	100	Conf 7
20	UT or ST	High	Low	80	Conf 8
21	UT or ST	Low	High	60	Conf 8
22	RespT	High	High	60	Conf 8
23	RespT	Low	High	100	Conf 8
24	RespT	Low	Low	60	Conf 8
25	UT or ST	High	Low	60	Conf 10
26	UT or ST	High	High	60 or 80 or 100	Conf 11
27	UT or ST	Low	Low	60 or 80	Conf 11
28	ReacT	High	High	80	Conf 11
29	ReacT or RespT	High	Low	60	Conf 11
30	SD	Low	High	60	Conf 11
31	ReacT or SD	Low	High	100	Conf 12
32	SD	High	High	80 or 100	Conf 12
33	SD	Low	High	80	Conf 12

The backfilling scheduling algorithm needs an estimated execution time for all submitted jobs as an input parameter [17]. As described before, the RGSA also needs input parameters, but these ones don't need to be introduced by each user (per job). Using past information (log files etc.), depending on the day and time, we can classify or divide workloads in groups (sub-workloads) in a time interval by the predominance level of a job type. For example, in Fig. 3, on Mondays between 0 a.m. to 6 a.m., based on a hypothetical log file, we noted that all executed jobs (predominance level equal to 100%) have a high execution time and high parallelism degree (HH100%). And in this period (night), the most important metric is utilization. So, according to our CCL implementation, the RGSA reconfigures to the configuration 11.

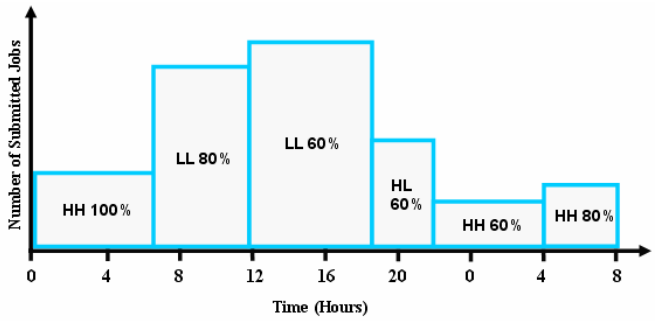


Fig. 3. The classification of a log file in sub-workloads, along the time, by predominance level of a job type.

This classification process can be done by a system administrator or an automated system that examines log files and classifies into sub-workloads. Along the time, the CCL table can be updated. As suggested in [9], the RGSA can use idle cycles to simulate the last executed sub-workload with all possible different configurations and update the table with the best configuration for this sub-workload. As we know, some system's behaviors repeat over the time. For example, if on last Monday at night, the RGSA found that configuration 11 was the best one, probably this configuration will achieve a good performance if RGSA uses it in the next Monday at night.

The selection of the most important metric can be done according to the predominance level of interactive and batch jobs in a workload. For interactive jobs, reaction time and response time are generally most important, because users want a quick answer. And for batch jobs, utilization is the most important, because the system administrator needs to use the maximum of the system resources. The definition of thresholds between high and low execution time and parallelism degree must be determined according to each system.

3 Related Work

This paper presents the main results of a master's thesis [14]. In this research, we found many works about gang scheduling [1], [2], [5], [6], [11], [12], [14], [18], [19],

[20], few works about reconfigurable software [13], [14], [16] and algorithms, and none about reconfigurable parallel job scheduling algorithms. Even so, all related works are deeply discussed in [14] and really helped us to reach our objectives and goals. In this paper, we will discuss only four papers that are more relevant and close to our work [6], [9], [12], [17].

In [12], a flexible co-scheduling algorithm is proposed and implemented. As well as our proposal, it uses a different algorithm depending on the workload. The gang scheduling is only used with jobs that really need it, while other jobs can be scheduled with no restrictions. This approach is limited to a couple of scheduling options. Moreover, the used gang scheduling algorithm is the traditional one.

Regarding the experimental results, [6] is the work that presents the closest experimental results to our research. By simulation, Feitelson compares many different packing schemes and few re-packing schemes, looking for the one which best performs on average for the used workload. Thus he does not present the idea that the same algorithm can assume different configurations, by changing its packing schemes, for example. Moreover he does not vary others parameters like the multiprogramming level and queue policies. Even so, it is very important to compare that work with some results that were achieved in our simulations.

In [9], Feiltelson presents the idea of self-tuning systems, in which the process to tune the system is automated. It uses genetic algorithms and log files as input for simulations. These simulations are performed during idle cycles, increasing the utilization of the system, with no cost.

Finally, in [17], a self-tuning job scheduler with dynamic policy switching is simulated and analyzed using trace information from some computing centers. Like backfilling schedulers it needs information about the job's estimated execution time. It is limited to three policies and conservative backfilling. It presents a fine idea of self-tuning that can be used in our Configuration Control Layer to change configurations.

4 EXPERIMENTAL METHOD

In this section, we first describe the metrics, parallel architecture and workload used in our simulations. Afterwards, we describe the experimental design in which we highlight the used configurations.

4.1 Metrics

In order to analyze a parallel job scheduling algorithm, we can use different metrics. The most common are: utilization, response time, reaction time and slowdown [7], [8], [15].

The mean utilization of a parallel architecture may be calculated through Eq.1, where *CPUBusyTime* is the time in which a processor was busy and *TotalTime* is the total time involved in the execution of all the workload. The utilization value is always between 0 and 1. The utilization depends directly on the input load. To compare different job scheduling algorithms under the same load and workload, the relative difference of the utilization is an important parameter to evaluate the obtained performance.

$$MeanUtilization = \frac{\sum CPUBusyTime}{NumberOfProcessors \times TotalTime} \quad (1)$$

The mean job response time (in seconds), defined in Eq.2, is the mean time interval between the submission and end of a job.

$$MeanResponseTime = \frac{\sum (JobEndTime - JobSubmissionTime)}{NumberOfJobs} \quad (2)$$

The mean job reaction time (in seconds), defined in Eq.3, is the mean time interval between the submission and the start of a job.

$$MeanReactionTime = \frac{\sum (JobStartTime - JobSubmissionTime)}{NumberOfJobs} \quad (3)$$

As shown in Eq.4, the mean jobs slowdown is the sum of jobs response times (reaction time + execution time) divided by the jobs execution times (dedicated time). This metric emerges as a solution to normalize the high variation of the jobs response time. The nearest the value is from 1, the better is the slowdown.

$$MeanSlowdown = \frac{\sum \frac{JobResponseTime}{JobExecutionTime}}{NumberOfJobs} \quad (4)$$

We decided to use the mean simulation time of the workload as a metric too, which is the time interval between the beginning and the end of the simulation (when the last job ends).

4.2 Parallel Architecture

The selected parallel architecture is a cluster composed of 16 nodes and a front-end node interconnected by a Fast Ethernet switch. Each node has a Pentium III 1 Ghz (real frequency = 0.938Ghz) processor. In Table 1, we see the main values of the cluster's characteristics, obtained from benchmarks and performance libraries (Sandra 2003, PAPI 2.3 etc.). These values are essential as input parameters to ClusterSim, a simulation tool developed by our group.

The ClusterSim is a Java-based parallel discrete-event simulation tool for cluster computing. It supports visual modeling and simulation of clusters and their workloads for performance analysis. In the simulation model, a cluster is composed of single or multi-processed nodes, parallel job schedulers, network topologies and technologies. A workload is represented by users that submit jobs composed of tasks described by probability distributions and their internal structure (CPU, I/O and communication instructions). The simulation model supports a lot of events: job arrival, end of job, unblock task, end of task, message arrival etc. For that reason, depending on cluster size and especially on the number of jobs, the execution of a simulation can be too long and the simulation tool can become out of memory [14].

Table 2. Cluster characteristics and respective values.

Characteristic	Value
Number of Processors	16 + 1
Processor Frequency	0.938 Ghz
Cycles per Instruction	0.9997105
Primary Memory Transfer Rate	11.146 MB/s
Secondary Memory Transfer Rate	23.0 MB/s
Network	Fast Ethernet
Network Latency	0.000179 s
Max. Segment Size	1460 bytes
Network Bandwidth (Max. Throughput)	11.0516 MB/s
Protocol Overhead	58 bytes

4.3 Workload

As described before, in our simulation tool, a workload is composed of a set of jobs featured by: their types, internal structures, submission probabilities and inter-arrival distributions. Due to the lack of information about the internal structure of the jobs, we decided to create a synthetic set of jobs [8], [10], [15].

In the related works [2], [5], [6], [10], [19], we found only information about the execution time of the jobs, but our simulation tool simulates a job execution based on its number of instructions. So we performed some pilot tests to define some of these values (number of instructions, granularity etc.) for our synthetic jobs. In order to simplify our jobs internal structures, we fixed some of the values and characteristics (Table 3).

In the workload jobs, at each one of the iterations, the master task sends a different message to each slave task. On their turn, they process a certain number of instructions, according to the previously defined granularity, and then they return a message to the master task. The total number of instructions that is to be processed by the job and the size of the messages are divided among the slave tasks, that is, the greater is the number of tasks (high parallelism degree) the least is the number of instructions that a single task has to process.

With regard to the parallelism level, which is represented by a probability distribution, we considered jobs between 1 and 4 tasks as low parallelism degree and between 5 and 16 as high parallelism degree. As we know, real workload analyses show that for large parallel machines (bigger than 64 processors), there are more small jobs. In our case, we did a relative equivalence. For example, in a 128-processors machine, short jobs are less than 32 tasks (one quarter). So, for a 16-processor machine, we considered a short job as less than 4 tasks (one quarter). As usual, we used a uniform distribution to represent the parallelism level, another more realistic way could be the use of a uniform distribution that samples power of 2 numbers. Combining the parallelism level, number of instructions and granularity characteristics, we had 8 different basic job types.

There are two main aspects through which a job can influence in a gang scheduling: space and time [7]. In our case, space is related with the parallelism degree and time with the: number of instructions, granularity and the other factors. Combining space (parallelism degree) and time (execution time), we can cover the majority of possible workloads. So, after the simulation, we can identify, in a log file, sub-workloads that fit into any of these combinations. Thus we combine these orthogonal aspects to form 4 workload types.

Table 3. Workload characteristics and their values.

Characteristic	Value
Granularity	Low – 1 million instructions High – 10 million instructions
Number of Instructions	Low – 100 million instructions High – 1 billion instructions
Parallelism Degree	Low – uniform distribution (1,4) High – uniform distribution (5,16)
Parallel Algorithm Model	Process Farm (Master Slave)
Message Size	16 Kbytes

In the first type, the most predominant are the jobs with a high parallelism degree and a structure that leads to a high execution time. In the second type, jobs with a high parallelism level and a low execution time predominate. The third one has the majority of jobs with a low parallelism degree and a high execution time. In the last workload, jobs with a low parallelism degree and a low execution time prevail. For each workload we varied the predominance level between 60%, 80% and 100% (homogeneous). For example, a workload HH60 is a workload composed of 60% jobs with a high execution time and a high parallelism degree, and the other 40% is composed of the opposite workload (low execution time and parallelism degree). So, we created 12 workloads to test the gang scheduling algorithms: HH60, HH80 and HH100;

HL60, HL80 and HL100; LH60, LH80 and LH100; LL60, LL80 and LL100.

In all workloads we use a total number of jobs equal to 100 (due to the ClusterSim simulation time and memory limitations) and the inter-arrival represented by an Erlang hyper-exponential distribution. To simulate a heavy load, we divided the inter-arrival time by a load factor equal to 100.

4.4 Experimental Design

It is important to note that each RGSA configuration is a traditional gang scheduling algorithm (TGSA). Because in a TGSA, its parts are fixed and cannot be changed over time. For example, in Table 4, Conf01 has the first fit, alternative scheduling, limited multiprogramming level and FCFS, and it cannot changes over time. Through the rest of this paper, TGSA and configuration will be treated as synonyms.

In order to test and analyze the performance of the RGSA, we used a full factorial model. A configuration of RGSA or a traditional gang scheduling algorithm is composed of a packing scheme, a re-packing scheme, a multiprogramming level and a queue policy. In Table 4, we observe the possible configurations of RGSA. The multiprogramming level was limited in 3. When the multiprogramming level is unlimited, it does not make sense to use a wait queue. Because, as soon as a job arrives, it will always fit into the matrix.

Table 4. RGSA configurations composed of packing and re-packing schemes, mulitprogramming levels and queue policies.

Configs	Mult. Level	Queue Policy	Packing Scheme	Re-Packing Scheme
Conf 01	Limited	FCFS	First Fit	Alternative Scheduling
Conf 02	Limited	SJF	First Fit	Alternative Scheduling
Conf 03	Limited	FCFS	First Fit	Slot Unification
Conf 04	Limited	SJF	First Fit	Slot Unification
Conf 05	Unlimited	X	First Fit	Alternative Scheduling
Conf 06	Unlimited	X	First Fit	Slot Unification
Conf 07	Limited	FCFS	Best Fit	Alternative Scheduling
Conf 08	Limited	SJF	Best Fit	Alternative Scheduling
Conf 09	Limited	FCFS	Best Fit	Slot Unification
Conf 10	Limited	SJF	Best Fit	Slot Unification
Conf 11	Unlimited	X	Best Fit	Alternative Scheduling
Conf 12	Unlimited	X	Best Fit	Slot Unification

Each one of the 12 configurations was tested with each workload, using 10 different simulation seeds. The selected seeds were: 51, 173, 19, 531, 211, 739, 413, 967, 733 and 13. So we made a total of 1440 (12 configurations X 12 workloads X 10 seeds) simulations.

5 Experimental Results

In this section, we present and analyze the performance of RGSA. First, for each metric, we present the results obtained by simulation and analyze the performance and influence of every frame. To do it, we compare sets of configurations in which the analyzed frame is filled out with different blocks and the other frames have a fixed block. At the end of this section, we compare between the performance of RGSA and every configuration individually.

5.1 Utilization

In Fig. 4, we present the relative mean utilization of the cluster among each configuration for all workloads. Considering the packing schemes (Fig. 5(a)), when the multiprogramming level is unlimited, the first fit provides higher utilization for HL and LH workloads.

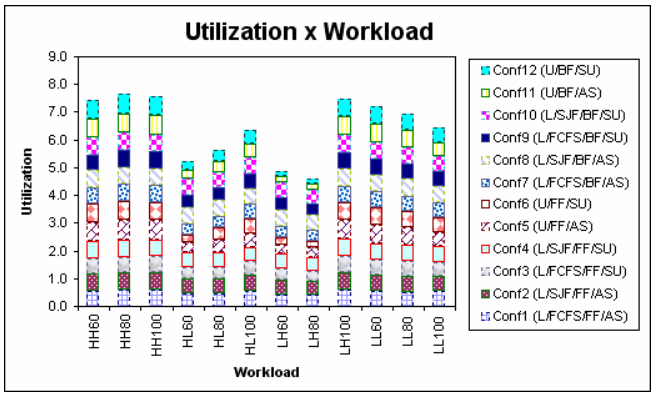
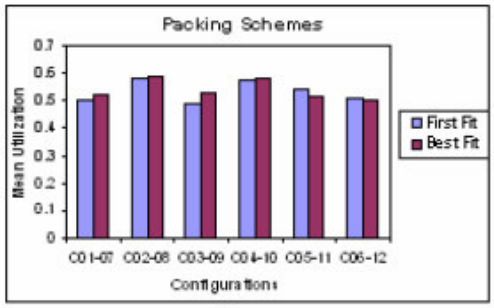


Fig. 4. The relative mean utilization among each configuration for all workloads.

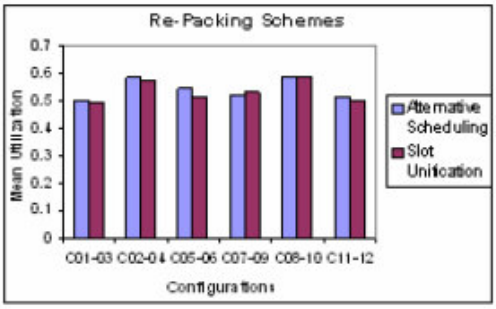
Initially, the best fit scheme finds the best slot for a job, but at long term, this decision may prevent new jobs from entering in more appropriate positions. In the case of HL and LH workloads, this chance increases, because the long jobs (with a low parallelism degree) that remain after the execution of short jobs (with a high parallelism degree) will probably occupy columns in common, thus, making it difficult to defragment the matrix. On the other hand, the first fit initially makes the matrix more fragmented. Besides, it increases the multiprogramming level. But at long term, it will make it easier to defragment the matrix, because the jobs will have fewer time slot columns in common. In the other cases, the best fit scheme presents a slightly better performance. In general, both packing schemes have a quite similar performance. The same happens to the re-packing schemes (Fig. 5 (b)).

Regarding the multiprogramming level, we reached two conclusions: the unlimited is better for HH and LL workloads (Fig. 5 (c)), but it is very bad for HL and LH workloads (Fig. 5 (d)). With an unlimited multiprogramming level, for each new job that does not fit into the matrix, a new time slot is created. At the end of the simulation, as the load is high, a large number of time slots existed. In this case, the big jobs (high parallelism level) are

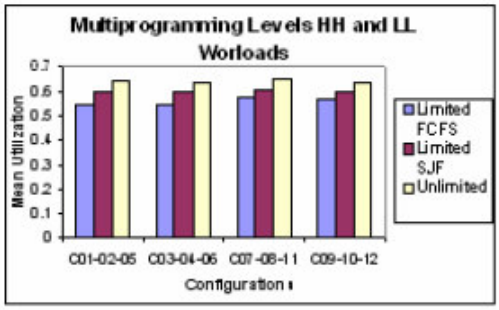
the long ones. So when the small jobs terminate, the idle space is significantly smaller than the space occupied by the big jobs, that is, the fragmentation is low and the utilization is maximized.



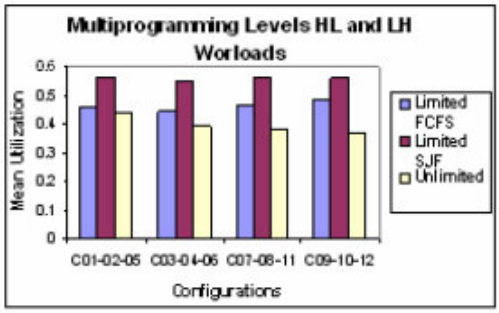
(a)



(b)



(c)



(d)

Fig. 5. Mean utilization considering the (a) packing schemes; (b) re-packing schemes; multiprogramming level for (c) HH and LL workloads; and (d) HL and LH workloads.

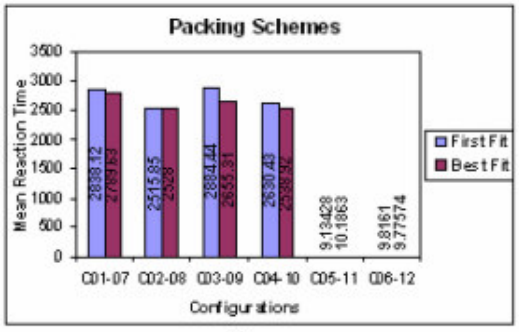
When we use LH and HL workloads, each matrix slot will be occupied by long and short jobs. As time goes by, the short jobs will end, leaving idle spaces on the matrix. In this case, the big jobs can not be the long ones, so a big space can become idle. Even if we use re-packing schemes, the fragmentation becomes high.

With reference to the queue policies, the SJF policy presented a higher utilization in all cases. When we remove the short jobs first, there is a higher probability that short idle slots exist where they can fit. Using the FCFS policy, if the first job is a big one, it can not fit into the matrix, thus, preventing other short jobs from being executed. So some slots become idle and the utilization low.

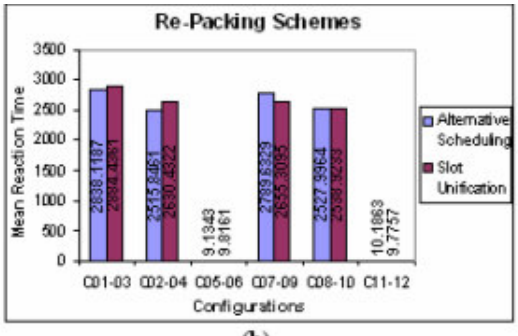
5.2 Reaction Time

In Fig. 7 we present the relative mean reaction time of jobs among each configuration for all workloads. Packing schemes have a very small influence on reaction time, because they depend on the new job that came from the wait queue. According to Fig. 6 (a), we can say that the both packing schemes are quite similar. The same happens to the re-packing schemes, because the defragmentation occurs after the beginning of the job's execution (Fig. 6 (b)).

The multiprogramming level has a direct influence on the jobs reaction time, because with an unlimited number of slots, a job can always fit into the matrix without waiting in the queue. In the worst case, the reaction time of a job will be equal to the number of slots multiplied by the slot quantum.



(a)



(b)

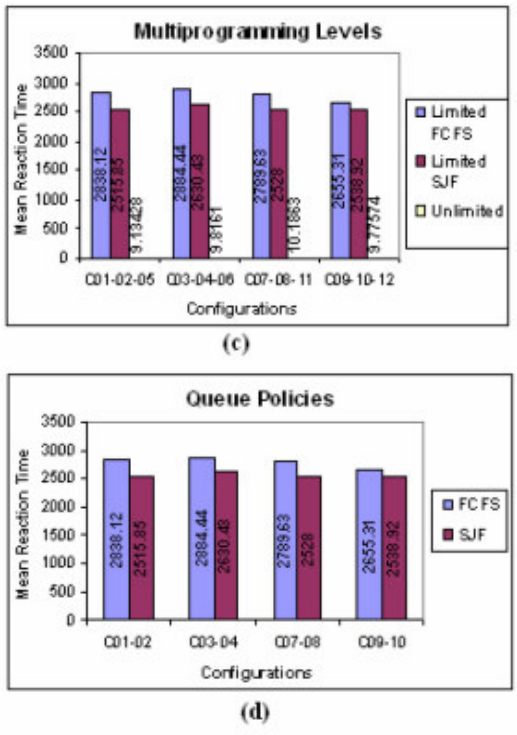


Fig. 6. Mean jobs reaction time considering the (a) packing schemes; (b) re-packing schemes; (c) multiprogramming levels; (d) queue policies.

Configurations with an unlimited multiprogramming level present an insignificant reaction time in comparison with those with a limited multiprogramming level, as shown in Fig. 6 (c).

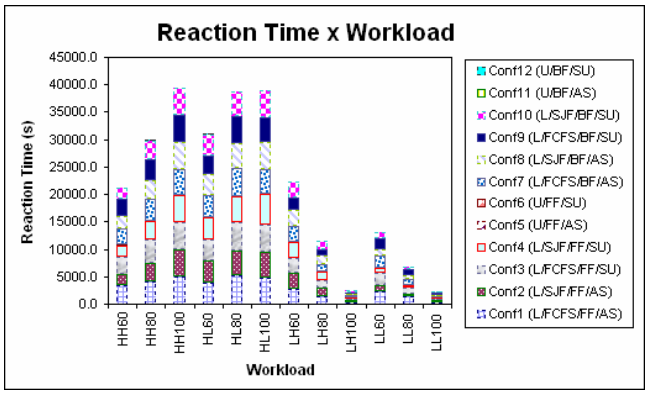


Fig. 7. The relative mean jobs reaction time among each configuration for all workloads.

With reference to the queue policies, on average, the SJF is better than FCFS, because the jobs in the queue spend less time waiting to be removed to the matrix and start their execution (Fig. 6 (d)).

5.3 Response Time

In Fig. 8 we present the relative mean jobs response time among each configuration for all workloads. According Fig. 9 (a) and (b), the results showed that both packing and re-packing schemes are equivalent. The multiprogramming level has a direct influence on the response time.

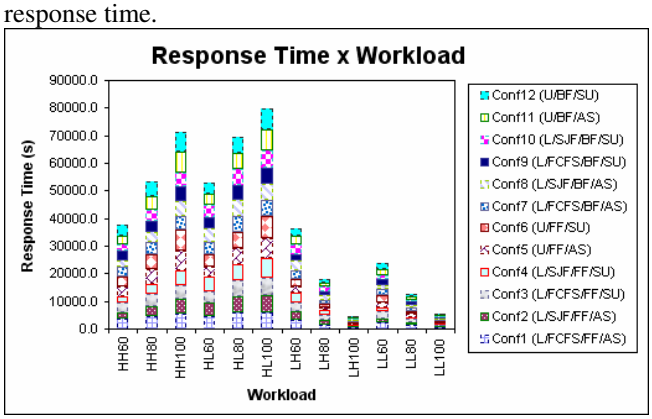
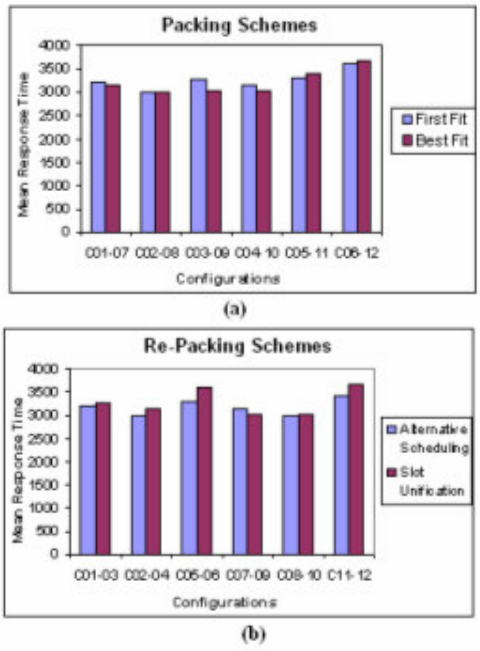


Fig. 8. The relative mean jobs response time among each configuration for and all workloads.

When the multiprogramming level is unlimited, the jobs have a short reaction time, but the execution time tends to be higher, because there are more available time slots (providing more concurrency). The execution time of a job is increased by the reaction time if the multiprogramming level was limited. On average, configurations with an unlimited multiprogramming level are worse than those a with limited one, that is, more jobs concurring in the matrix is worse than more jobs waiting in the queue, but there are some exceptions.



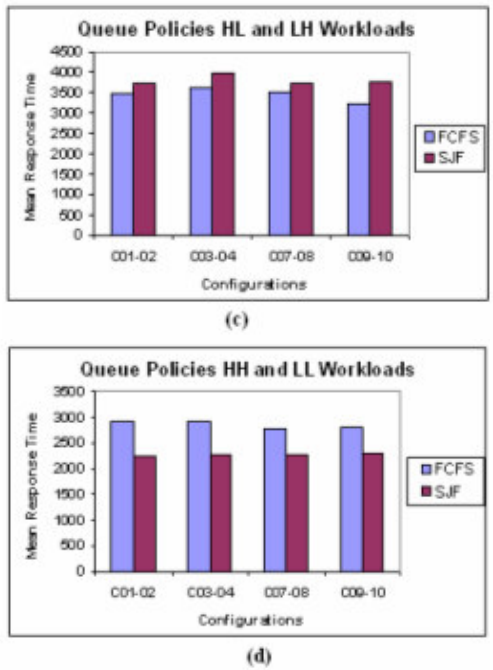


Fig. 9. Mean response time of jobs considering the (a) packing schemes; (b) re-packing schemes; queue policies for (c) HL and LH workloads and (d) HH and LL workloads.

Generally, we believe that unlimited multiprogramming is always better if we are not considering memory paging, but in Fig 10, we see a simple example in which the mean jobs response time is better (smaller) for a limited multiprogramming level. Suppose a workload composed of three jobs with 2 tasks (each one) and an execution time equal to 2.1 seconds; and an Ousterhout matrix with two columns, a time slice equal to 1 and a limited multiprogramming level equal to 2. In this example, when a job finishes before the time slice ends, a new time slice starts.

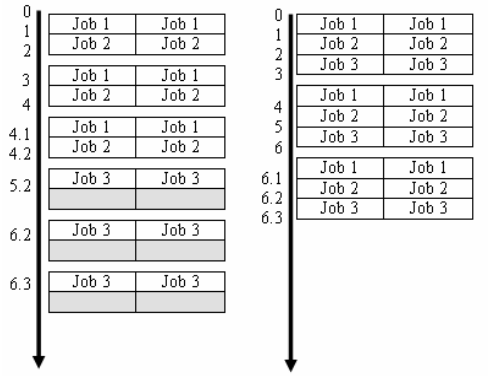


Fig. 10. The simple workload execution using limited and unlimited multiprogramming levels.

In Fig. 10, we observe the following response times for limited multiprogramming level: Job1 = 4.1s; Job2 = 4.2s; Job3 = 6.3s; mean = 4.86s. And for unlimited multiprogramming level we observe the following response times: Job1 = 6.1s; Job2 = 6.2s; Job3 = 6.3s; mean = 6.2s. So, we note that the use of a limited multiprogramming level can achieve better response times for a certain workload, even not considering the memory paging.

With reference to the queue policies, we reached two conclusions: the SJF policy is better for HH and LL workloads (Fig. 9 (c)) and the FCFS policy is better for HL and LH workloads (Fig. 9 (d)). In the first case, the LL jobs are initially executed and terminated quickly. Thus, HH jobs wait less time in the wait queue, reducing their reaction time and consequently their response time.

In the last case, when we use the SJF policy, the HL jobs are executed first. So LH jobs have to wait so much time in the queue, which increases their reaction time and consequently their response time.

5.4 Slowdown

In Fig. 11 we present the relative mean jobs slowdown among each configuration for all workloads. Based on past analysis, we conclude that both the packing and re-packing schemes are equivalent. The slowdown is based on response time and consequently on reaction time. When the multiprogramming level is unlimited, the response time is almost equal to the execution time. Thus, the slowdown value tends to 1.

Based on the reaction time analysis and the results shown in Fig. 9, on average, the SJF policy presents a better slowdown, but there are exceptions.

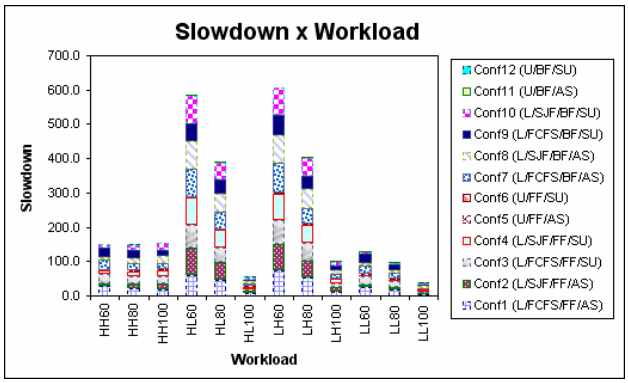


Fig. 11. The relative mean jobs slowdown among each configuration for all workloads.

5.5 Simulation Time

In Fig. 12, we present the relative mean simulation time among each configuration for all workloads. The simulation time depends directly on the utilization. So, all observations and analyses of the utilization metric may be extended to the simulation time.

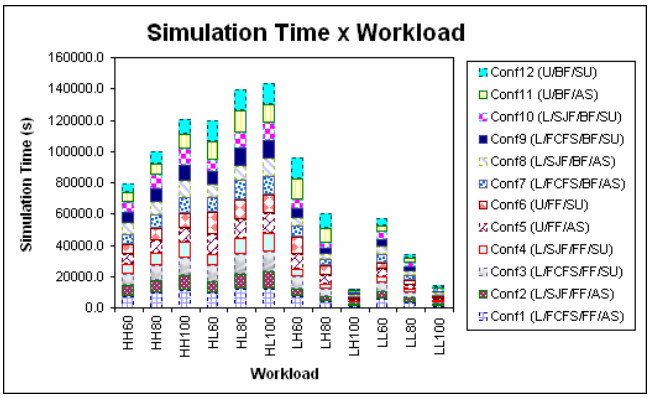


Fig. 12. The relative mean simulation time among each configuration for all workloads.

5.6 RGSA Analysis

In order to analyze the performance of RGSA, we need to compare it to each configuration or traditional gang scheduling algorithm individually. As we said in the proposal of RGSA, based on log files, the workload is divided in sub-workloads that fit into one of the workload classes (ex: LL100%). Then the CCL evaluates the entry parameters, reconfiguring RGSA to the best configuration. As we are not using a trace, we analyze RGSA for all proposed workloads.

In Table 5, we observe that on average, considering all metrics, RGSA is 36.61% better than the 12 traditional gang scheduling algorithms. Note that if we had chosen Conf5 (the best configuration on average), RGSA would still be 11.45% better.

Table 5. Speedup, in percentage (%), of the RGSA performace when compared to each configuration for a workload composed of 12 described workloads.

Metrics / Configurations	Utilization	Reaction Time	Response Time	Slowdown	Simulation Time	Mean
Conf01	18.8153	99.6937	18.1988	96.6667	19.4711	**50.5691**
Conf02	5.5793	99.6545	12.7879	96.3408	5.3772	**43.9479**
Conf03	20.2381	99.6986	20.1357	96.6585	21.5047	**51.6471**
Conf04	7.1421	99.6695	16.6365	96.4005	8.1059	**45.5909**
Conf05	12.2866	4.8307	21.1611	0.0566	18.9160	**11.4502**
Conf06	17.0448	11.4411	27.6418	0.1415	25.2068	**16.2952**
Conf07	16.0706	99.6884	16.7549	96.9840	17.8183	**49.4632**
Conf08	5.1443	99.6561	12.9742	96.4882	5.2786	**43.9083**
Conf09	14.6709	99.6726	13.7521	96.3212	15.1375	**47.9109**
Conf10	6.0941	99.6576	13.6044	96.4374	5.8391	**44.3265**
Conf11	16.6605	14.6595	23.2316	0.2179	27.7065	**16.4952**
Conf12	19.0915	11.0755	28.8892	0.0591	29.9556	**17.8142**
Mean	**13.2365**	**69.9498**	**18.8140**	**64.3977**	**16.6931**	**36.6182**

Now, we analyze another example, in which the workload is composed of HL60 and LH60 workloads. According to Table 6, on average, the speedup of RGSA increases to 41.53%, and with reference to Conf5, this speedup increases to 18.83 considering all 5 metrics. If we consider only the utilization metric, the speedup of RGSA over Conf5 increases from 18.83% to 42.32%. In the last case, we note that Conf5 (the best on average) would be worse than Conf3, which was previously considered the worst configuration.

Table 6. Speedup of RGSA, in percentage (%), when compared to each configuration for a workload composed of HL60 and LH60 workloads.

Metrics / Configurations	Utilization	Reaction Time	Response Time	Slowdown	Simulation Time	Mean
Conf01	26.7836	99.7605	22.5123	98.4916	27.1878	**54.9471**
Conf02	0.8016	99.7739	31.9511	98.6920	0.7671	**46.3972**
Conf03	**29.0247**	99.7647	24.4469	98.4859	29.0702	**56.1585**
Conf04	4.2309	99.7809	34.1388	98.7164	4.3209	**48.2376**
Conf05	**42.3255**	4.0597	3.8764	0.0324	43.8949	**18.8378**
Conf06	50.3595	5.2884	19.6931	0.0742	51.3580	**25.3546**
Conf07	28.2399	99.7709	25.0613	98.7869	28.8798	**56.1478**
Conf08	0.7319	99.7739	31.6864	98.7537	0.8703	**46.3632**
Conf09	19.0918	99.7198	12.7145	98.1925	18.9861	**49.7409**
Conf10	1.0836	99.7729	31.9171	98.7256	0.8443	**46.4687**
Conf11	52.1549	0.0000	10.7741	0.0984	53.2991	**23.2653**
Conf12	54.4356	0.7533	21.7874	0.1247	55.1504	**26.4503**
Mean	**25.7720**	**67.3516**	**22.5466**	**65.7645**	**26.2191**	**41.5307**

With these examples, we show that the use of reconfiguration concepts in gang scheduling algorithms may provide a high speedup over traditional gang scheduling algorithms.

In these examples, we considered that there weren't reconfiguration overheads, neither wrong workload classifications nor classification overheads. The reconfiguration overhead is insignificant, it is just the time spent to execute a switch case structure (to select the more appropriated configuration) and fit some specific blocks in the frames to change the configuration. The classification of the workload based on log files in sub-workloads and the CCL table's update can be done using idle cycles. But how to classify the workload and the consequences of wrong classifications are open and interesting topics to a more detailed research. In despite of these overheads and costs, the speedup may be great enough to make RGSA a good alternative.

6 Conclusion

In this paper, we defined, proposed, developed, implemented (in a simulation tool) and analyzed the performance of RGSA by simulation. As general conclusions about the RGSA frames, we can highlight:

1. Packing Schemes Frame. Considering all metrics, on average, both packing schemes (first fit and best fit) presented an equivalent performance, as found in [6]. It suggests that other constructive blocks may be used.

2. Re-Packing Schemes Frame. Considering all metrics, on average, both re-packing schemes (slot unification and alternative scheduling) presented an equivalent

performance. It suggests that other constructive blocks may be used.

3. Multiprogramming Levels Frame. Considering the metrics: utilization and simulation time, the unlimited multiprogramming level presented a better performance to HH and LL workloads, and the limited one for the HL and LH workloads. For reaction time and slowdown metrics, the unlimited multiprogramming level presented a best performance in all cases. Finally, considering the response time metric, on average, the limited multiprogramming level was the best.

4. Queue Policies Frame. Considering the utilization and simulation time metrics, the SJF policy was always better than the FCFS. For reaction time and slowdown metrics, on average, the SJF policy presented a better performance, but in some specific cases FCFS was better than the other. Finally, considering the response time metric, the SJF policy presented a better performance for the HH and LL workloads and the FCFS policy for the HL and LH workloads.

On average, the performance of RGSA was around 40% better than the other traditional gang scheduling algorithms for all tested workloads. One of the most important results was to show that depending on the selected metric and workload, the best algorithm on average for all situations (Conf5) may be worse than the worst algorithm on average (Conf3). In our simulations, the performance of RGSA was 42.32% better than the one of Conf5 in a specific situation. So, the use of a reconfigurable algorithm may largely improve the system's performance.

In our specific case, the longest simulation took about 13000 seconds (3 hours and 36 minutes). So we got to reduce the simulation time in 40% (1 hour and 26 minutes) using RGSA. But in real systems, a workload may execute for a week. In that case, a reduction of 40% would mean to reduce the workload execution time in 2.8 days

In this paper, we proposed a model or architecture of a reconfigurable algorithm that was applied in gang scheduling. But this model can be applied on any other scheduling algorithm. Using a reconfigurable algorithm, developers don't need to create a monolithic algorithm based on behavior description that works well for all situations. They may design a set of structural algorithms or parts of an algorithm that everyone is optimized for a different situation.

The **main contributions** of this paper are: the definition, proposal, implementation and performance analysis of RGSA, comparing it with other traditional gang scheduling algorithms for different workloads.

As **future works** we can highlight: the inclusion of new frames and blocks in RGSA; an adaptive CCL; compare RGSA with backfilling schemes; study on how to classify the workload found in log files into sub-workloads; tests with other workloads, simulation with a bigger number of jobs, different loads, other jobs algorithm models and communication patterns, clusters of different sizes and tests in a real environment.

ACKNOWLEDGMENT

We would like to thank the Graduate Program in Electrical Engineering, Computational and Digital Systems Laboratory (LSDC), CAPES and ProPPG for the support.

REFERENCES

[1] Batat, A., Feitelson, D.: Gang Scheduling with Memory Considerations. IEEE International Parallel and Distributed Processing Symposium. (2000) 109-114

[2] Chapin, S.J. et all: Benchmarks and Standards for the Evaluation of Parallel Job Schedulers. Job Scheduling Strategies for Parallel Processing. (1999) 67-90

[3] Compton, K., Hauck, S.: Reconfigurable Computing: A Survey of Systems and Software. ACM Computing Survey. (2002)

[4] Dehon, A.: The Density Advantage of Configurable Computing. IEEE Computer, Vol. 33, No. 4. (2000)

[5] Feitelson, D., Rudolph, L.: Evaluation of Design Choices for Gang Scheduling using Distributed Hierarchical Control. Journal of Parallel & Distributed Computing (1996) 18-34

[6] Feitelson, D. G.: Packing Schemes for Gang Scheduling. Job Scheduling Strategies for Parallel Processing. (1996) 89-110

[7] Feitelson, D.G.: A Survey of Scheduling in Multiprogrammed Parallel Systems. Research Report RC 19790 (87657). IBM T. J. Watson Research Center (1997)

[8] Feitelson, D., Rudolph, L.: Metrics and Benchmarking for Parallel Job Scheduling. Job Scheduling Strategies for Parallel Processing. (1998) 1-24

[9] Feitelson, D. G. and Naaman,, M.: Self-tuning Systems. IEEE Software. (1999) 52-60

[10] Feitelson, D.: Metric and Workload Effects on Computer Systems Evaluation. IEEE Computer. (2003) 18-25

[11] Franke, H., Jann, J, Moreira, J., Pattnaik, P., Jette, M.: An Evaluation of Parallel Job Scheduling for ASCI Blue-Pacific. ACM/IEEE Conference on Supercomputing. (1999)

[12] Frachtenberg, E., Feitelson, D.G., Petrini, F. and Fernandez, J.: Flexible CoScheduling: Mitigating Load Imbalance and Improving Utilization of Heterogeneous Resources. 17th International Parallel and Distributed Processing Symposium. (2003)

[13] Góes, L. F. W., Martins, C. A. P. S.: RJSSim: A Reconfigurable Job Scheduling Simulator for Parallel Processing Learning. 33rd ASEE/IEEE Frontiers in Education Conference. Colorado (2003)

[14] Góes, L. F. W., Martins, C. A. P. S.: Proposal and Development of a Reconfigurable Parallel Job Scheduling Algorithm. Master's Thesis. Belo Horizonte, Brazil (2004) (in portuguese)

[15] Jann, J., Pattnaik, P. and Franke, H.: Modeling of Workload in MPP's. Job Scheduling Strategies for Parallel Processing. (1997) 95-116

[16] Martins, C. A. P. S., Ordonez, E. D. M., Corrêa, J. B. T., Carvalho, M. B.: Reconfigurable Computing: Concepts, Tendencies and Applications. SBC JAI - Journey of Actualization in Informatics. (2003) (in portuguese)

[17] Streit, A.: A Self-Tuning Job Scheduler Family with Dynamic Policy Switching. 8th WJSSPP, Springer Verlag LNCS Vol 2537. (2002) 1-23

[18] Wiseman, Y., Feitelson, D.: Paired Gang Scheduling. IEEE Transactions Parallel and Distributed Systems. (2003) 581-592

[19] Zhang, Y., H. Franke, Moreira, E.J., Sivasubramaniam, A.: Improving Parallel Job Scheduling by Combining Gang Scheduling and Backfilling Techniques. IEEE International Parallel and Distributed Processing Symposium. (2000)

[20] Zhou, B. B., Brent, R. P.: Gang Scheduling with a Queue for Large Jobs. IEEE International Parallel and Distributed Processing Symposium. (2001)