Analysis of Scheduling Algorithms with Migration Strategies in Distributed Systems

Francisca Aparecida P. Pinto*, Chesley B. Chaves†, Lucas G. Leite‡, Francisco Herbert L. Vasconcelos§ and Giovanni C. Barroso¶

Federal University of Ceará (UFC)
Department of Teleinformatics Engineering *§¶, Department of Computer Science‡ and UFC Virtual Institute†§, Ceará, Brazil
{aparecida.prado, herbert}@virtual.ufc.br, chesleybraga@gmail.com, lucasgml@alu.ufc.br, gcb@fisica.ufc.br

Abstract—**Task scheduling is a problem which seeks to allocate, over time, various tasks from different resources. In this paper, we consider group task scheduling upon a heterogeneous multi-cluster system. Two types of job tasking are considered, parallel and sequential. In order to reduce fragmentation caused by the scheduler group, migration mechanisms were implemented. Moreover, the dispatchers global and local use distribution of jobs in order to minimize delays in the task queues, as well as in response time. To analyze the different situations, performance metrics were applied, aiming to compare schedulers in different situations.**

Keywords-parallel job; scheduling; distributed Systems;

I. Introduction

Traditionally, the main focus of the industry has been to improve the performance of computing systems through more efficient projects, hereby increasing the density of its components. Associated with the exponential growth of data size in simulation/scientific instrumentation, storage and internet publications, the increased computational power of such systems boosted investment by big providers, government research laboratories and computing environments, thus enhancing robustness in order to host applications ranging from social networks to scientific workflows [1].

In this context, distributed systems emerge as an interesting solution for the provision of physical resources upon demand, as they allow additional computational power of several nodes interconnected by a computer network, in order to perform tasks. Distributed computing systems have been used due to their important attributes, such as: cost efficiency, scalability, performance and reliability. In grid computing, there are three important aspects that must be treated: task management, task scheduling and resource management [2]. In particular, Grid Task Scheduling (GTS) plays an important role in the system as a whole, and its algorithms have a direct effect on the grid system. Task scheduling in a heterogeneous computing environment proved to be a NP-complete problem [3].

To solve this problem, various scheduling algorithms have been proposed for distributed environments, whereby they have been classified in several different ways. For example, as shown in [4], we propose a hierarchical tree classification which splits at the highest hierarchy level into the local and global algorithms. With reference to [5], the authors classify the algorithms according to the types of applications found in the grids: meta-task and Directed Acyclic Graph (DAG) algorithms. Actions can be executed simultaneously and independently through meta-task algorithms, whereas the type DAG algorithm contains precedence constraints. Moreover, they are classified into traditional algorithms, deterministic and heuristic intelligence, for the use of different optimization technologies.

Existing scheduling techniques, highlight scheduling groups or co-schedulers [6], are considered efficient algorithms for the purpose of scheduling parallel jobs that consist of tasks that must be allocated and executed simultaneously on different processors [7]. These types of scheduling algorithms provide interactive response times for tasks with low execution time by means of preemption, with the disadvantage of causing fragmentation and reducing system performance [8]. The fragmentation of resources has been a common subject of research in the last two decades [16]. Various approaches to the fragmentation of resources have been developed, whereby better fit and task migration are the two most common approaches.

Based upon the above, this paper aims to reduce the fragmentation caused by the scheduler group and response time. Therefore, the main contributions of this work are: i) application of a migration mechanism task schedulers group, in order to minimize the fragmentation and response time. ii) implementation of strategies inside dispatcher managers, aiming the distribution of tasks to the clusters, to avoid unnecessary migrations, improving system efficiency; and iii) implementation of a heterogeneous multi-cluster system with the objective of analyzing the performance of schedulers in different situations, as well as the system behaviour in different contexts.

This paper is organized as follows: Section II presents related work; Section III presents the model of the proposed system; Section IV describes the operation of the system; Section V illustrates the group schedulers and mechanisms of migration; Section VI presents the performance metrics; Section VII presents the results of simulations, and finally, Section VIII covers the completion of the work, along with the prospects for future development.

II. Related Work

The group schedulers, Adapted First Come First Served (AFCFS) and Largest Gang First Served (LGFS), used in this work, have been studied in a distributed environment, [7][9][10][11]. The aim of this study is to make them more efficient in scheduling of parallel tasks, since these algorithms cause fragmentation in the system. In [7][10][11][12], there are proposed migration mechanisms, which are utilized in order to minimize the fragmentation caused by the schedulers group in the distributed environments. In this study, besides the technical migration, other strategies are used (Section V-A), in order to avoid unnecessary migrations, as well as overloading of the system. In [13][14], the authors propose a two levels system model within a grid environment. In the first level, containing the global scheduler, there is an overview of the application of the job task, and subsequently, the second level scheduler has knowledge of all resource details. Moreover, they consider load balancing through the global manager scheduler only. In our proposal, we implemented a model that uses heterogeneous multi-cluster system managers, Grid Dispatcher (GD) and Local Dispatcher (LD), for the purpose of allocating jobs on resources. Therefore, unlike the authors mentioned above, we introduce the GD before sending jobs to clusters, which is information feedback regarding the load of the clusters, so that a more efficient load balance is achieved. Moreover, the distribution of tasks for the processor queue, the LD, requests information regarding the load of each processor, namely the number of running tasks in a long processor queue. Such information is required in order to reduce the time in the task queue and the response time of a job.

In some work studies concerning the group schedulers above, migration mechanisms are used to reduce the fragmentation of the system, whereby a metric used to evaluate the scheduler in relation to fragmentation is not applied. Therefore, in this paper, in order to analyze the different applied situations in addition to other metrics, we use the Loss of Capacity (LoC) function, so that we are able to analyze the performance of schedulers in different situations, as well as the behaviour of the system in different contexts. The metric LoC is relevant to measure both the use of the system as the fragmentation [15][16][17]. These authors applied the metric in different contexts.

III. Environment Description

The simulation environment consists of a grid multi-cluster system which uses the hierarchical structure of two layers. Such an environment was developed by the Research Group in Applied Computational Modeling of the UFC, in Java. This system is composed of managers, Grid Dispatcher (GD) and Local Dispatcher (LD).

The GD is responsible for the sending of jobs, both parallel and non-parallel for clusters, and LD is responsible for sending the task to the jobs belonging to the ranks of processors based on the algorithm Join The Shortest Queue (JSQ). Each cluster is comprised of a LD and a set of processors. The system is heterogeneous in terms of clock rate *r* and the number of processors *p* per cluster. The clock on machines can vary between $1500 \leq r \leq 3000$ (megahertz), which is randomly generated upon creation of the resources in the simulation environment. More importantly, the larger the value of *r*, the time between each processing cycle will be shorter and therefore tasks are executed in less time. In the implemented system, the distribution processor *p* is made in two clusters of 32 and 64 processors respectively, whereby each processor has its own queue. The model system is illustrated in Fig. 1.

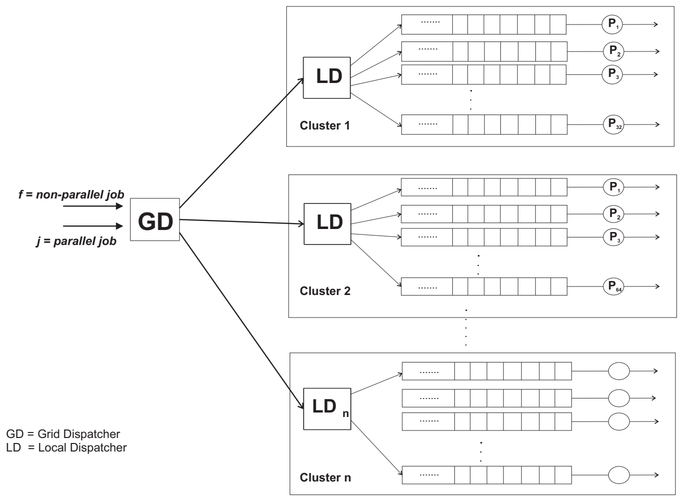

Figure 1. Multi-cluster system based on queues

In this environment, we assume that the two clusters belong to the administrative domain, such that they can communicate with GD. Moreover, communication between processors is contention free. Thus, we consider the communication latencies are implicitly included on the service time of the job. The workload applied in this system was extracted from a real distributed environment [18]. This workload is composed of two different kinds of jobs that are competing for the same resources: non-parallel job *f* and parallel job *j*. In the workload, each job is described by a tuple (*id*, *at*, s_j, *pt*): identification of job *id*, arrival time *at* where $at > 0$; s_j size of a job, in which $1 \leq s_j \leq 64$, and the processing time *pt*, whereby $pt > 0$.

In this paper, we assume that the value of a job *pt* workload will be applied in a machine, whereby $r \doteq 2000$ megahertz, as it currently has a median frequency processor. Otherwise, the *pt* undergoes change and may vary proportionally according to the clock of the machine, that is, when the value of $r \neq 2000$ megahertz (standard). Therefore, the calculation of the new processing time P_t is defined by (1).

$$P_t = pt \times \frac{C_d}{C_m} \qquad (1)$$

whereby *pt* is the processing time of the job on the workload, C_d is the standard clock where $C_d \doteq 2000$ (megahertz), and C_m represents the value of the *r* of the machine, which relates to $1500 \leq r \leq 3000$.

A job, *f*, consists of a single task, whose execution begins immediately upon its arrival on the grid. The system is not capable of responding to more than 64 jobs f per time unit. At present, a job *j* consists of t_j tasks, where $1 < t_j \leq 64$, hence, the number of tasks in a job *j* cannot exceed the number of processors in a cluster. Thus, the risk that a job may never be answered is null. Moreover, mapping between tasks and processors must be one to one. Thus, tasks from the same job cannot be attributed to the same processor queue. A job f is a high priority task, requiring only a processor *p* for its execution. Therefore, a processor that receives a priority task must immediately stop the execution of any other task type *j*, in order to serve the task type *f*. If a job *j* has one of its tasks interrupted by a job *f*, then each sibling task of *j* has to stop its execution, and then be rescheduled. Stopping the task t_j that belongs to a job *j*, can affect even more response time of *j*. As soon as f ends its execution, interrupted tasks can begin their execution again, as well as the entire process. It is noteworthy that disruption only occurs when all processors are busy. Moreover, jobs f can not interrupt one another.

IV. Operation of System Managers

This section will describe in detail, the operation of system managers, in other words, the Grid and Local Dispatchers, as illustrated in Fig. 1.

A. Grid Dispatcher

As stated earlier, the GD is responsible for sending jobs to the clusters. This submission is made based upon feedback information concerning the total load of each cluster, i.e., the total number of jobs in the queues, plus the number of tasks that are running on the processors. This cluster load information will be sent only at the request of GD, since excessive feedbacks may cause overload on the system. The average load between clusters is very important for more efficient load balancing. If the clusters are randomly balanced, then dispatch occurs. The calculation of the total load of the cluster is defined by (2).

$$C_i = \frac{1}{n} \times \sum_{p=1}^{n} [t(p) + t_s(p)] \tag{2}$$

whereby C_i is the total load per cluster, *n* is the total number of processors per cluster, *t(p)* is the total number of tasks in the queue of each processor *p*, and $t_s(p)$ represents the existence or non existence of a task running on processor *p*, $t_s(p) = 1$, if there is a running task, otherwise $t_s(p) \doteq 0$.

B. Local Dispatcher

After the parallel job *j* has been sent to a cluster *c*, according to the load of *c*, LD assigns the tasks to the available queues based on algorithm JSQ. JSQ is responsible for sending the tasks that belong to a job queue for processors that have fewer tasks in their own queues. It is important to emphasize that when a job *f* arrives at LD, it is forwarded to the cluster by JSQ.

In this work, an adaptation has been implemented in LD as a new criterion in the selection of the processor. This adaptation works as follows: LD receives information about the cluster load of each processor, i.e., the number of tasks in the processor queue plus the running task t_s. The information feedback only occurs when LD asks, thus avoiding system overload. The knowledge of the load of each processor is to minimize the delay of tasks to the processor queues and the response time of the job. The calculation of the adaptive LD, the adaptive Local Dispatcher (aLD) is defined by (3).

$$N_t(q) = n_t + t_s \tag{3}$$

whereby $N_t(q)$ is the total number of jobs per queue, q is the size of the processor queue, n_t represents the number of jobs in the queue and t_s represents the existence or non existence of a task running on processor, $t_s \doteq 1$ if there are any tasks running on the processor, otherwise, $t_s \doteq 0$.

With the aLD, JSQ sends tasks to the processors more efficiently, since this algorithm now has information of the effective value of the load of each processor. In Section VII, we present the impact that the adaption on LD causes on the results of the response time of the jobs. For this analysis, the LD will be applied with and without the adaptation in both group scheduling algorithms. In the next section, we present the group schedulers that were used for scheduling jobs in the queues of the processors.

V. Group Schedulers

In the simulation model, the following policies have been applied to the analysis of queues: AFCFS and LGFS [7][9] [10][11]. These schedulers were modified and implemented in each cluster, separately.

The algorithm AFCFS tends to favour jobs that consist of a number of smaller tasks, so that jobs that require a smaller number of processors, result in an increased response time for larger jobs. But the LGFS tends to favour the performance of bigger jobs instead of the smaller ones, i.e., bigger jobs have their jobs put on queues of processors before any others belonging to a job with a smaller size, resulting in an increase of response time of smaller jobs. In addition, LGFS involves a considerable amount of overhead in the system. Therefore, such scheduling algorithms cause fragmentation in the system, which happens in two stages: i) the schedulers cannot always meet the requirements of a job *j*, since the latter requires a number of available processors equal to the number of tasks, in order to execute; and ii) when there are idle nodes and tasks waiting in the queue to be executed, they are not able to schedule these tasks.

A. Migration

Assuming that the group scheduling causes fragmentation in the environment, we look at the use of migration to reduce fragmentation. In this work, we have studied different migration schemes for heterogeneous systems, in order to minimize such problems. Therefore, we assume two types of migration: local migration m_l and external migration m_e.

The difference between migration m_l and m_e is that the latter causes a higher overhead on the system, since it involves the transfer of tasks from one cluster to another. Therefore, the following strategies have been proposed in order to reduce fragmentation, as well as unnecessary migration, and consequent overloading of the the system:

1) checks all clusters that have processors available;
2) analyzes which jobs has its tasks at the beginning of the queue of idle processors;
3) based upon the above analysis, it checks which of the jobs has the least or equal number of tasks to that of the idle processors;
4) and finally transfers the tasks of the job that has fewer number of tasks to migrate.

During the migration tasks, the destination nodes are reserved, in order to prevent other tasks using them. When the target processor is reserved, we ensure the immediate start up of their performances after the migration of the tasks are chosen. The only way you can prevent the execution of these migrated tasks, is the arrival of a job *f*. If this problem occurs, migrated tasks are reserved, and when job *f* liberates the processor, they return and begin execution immediately. The reserved node can only be occupied by another job if it is of type *f*.

The reservation mechanism prevents elected tasks returning to the queue, because the scheduler AFCFS or LGFS would not be able to schedule such tasks. These schedulers are not suitable for parallel scheduling of tasks in different clusters. Eventually, the m_e was applied in an attempt to use more resources as to avoid resources remaining idle. In addition, we have the use of aging, in order to regulate the number of migrations that occur in the system, as well as reducing the starvation of tasks that came before.

The migration strategies were employed in the algorithms of AFCFS schedules and LGFS (Section V), which were used in each cluster separately. Therefore, these algorithms with migration are defined as AFCFSm and LGFSm, respectively. First, the scheduling hierarchy requires to run the scheduling algorithms AFCFS and LGFS, and then the migration m_l tries to schedule the jobs that were not able to be allocated by the scheduling algorithm. The migration m_e can only be used in an attempt to make use of more resources. The reason for this hierarchy is the overhead imposed by each of these steps. Unlike migration techniques, the schedulers (AFCFS and LGFS) without migration does not cause any additional overhead.

In the next section, we will describe the performance metrics applied to analyze the system model behaviour in different situations.

VI. Performance Metrics

In this work, we applied the following performance metrics: Average Response Time (ART), Utilization (U) and Loss of Capacity (LoC), which constitutes everything required in order to analyze the performance of schedulers in different situations, and the system behaviour in different contexts.

A. Average Response Time

The metric response time *rt* (in time units) measures the time interval between the arrival of the job in the system until the end of its execution [17], thus, the average response time or ART is given by (4).

$$ART = \frac{1}{m} \times \sum_{j=1}^{m} rt(j) \qquad (4)$$

whereby ART is the average response time of jobs, *rt(j)* represents the response time of a job, and *m* is the total number of jobs executed.

B. Utilization

In simulation studies, the metric used is simply an indirect measure of makespan [20], with the workload constant for all schedulers. The calculation of the metric used is given by (5):

$$U = \frac{\sum_{j=1}^{m} p_j \times te_j}{makespan \times N} \qquad (5)$$

whereby U is utilization of the clusters, p_j represents the number of processors that each job needs for its implementation, te_j represents the execution time of each job, N represents the size of the system and *m* is the total number of jobs executed.

C. Loss of Capacity

This metric is important for measuring both uses of the system as fragmentation. In a system, fragmentation occurs when: (i) there are tasks waiting in the queue to run; and ii), there are idle nodes, but still unable to run the waiting tasks. LoC reflects the costs of fragmentation. These metrics have been used in some of the works as detailed in [15][17][19], respectively. To use the LoC, we assume two factors: 1) the number of tasks of a job, *j*, can not exceed the number of processors in the system, avoiding the starvation, in other words, the job would never be serviced; 2) the variable δ (delta), (6), represents the state of the processors and jobs in the system. For example, $\delta = 1$ indicates the existence of enough available processors to perform at least one job in the queue at the moment a new job is scaled. Likewise, when $\delta = 0$, if the queues are empty or do not exist in the same job size less than or equal to the number of idle processors. The metric LoC is calculated as follows:

$$LoC = \frac{\sum_{j=1}^{q-1} n_i(t_{i+1} - t_i)\delta}{N(t_q - t_1)} \qquad (6)$$

whereby *q* represents the number of jobs in the staggered moment when a new job is scheduled or when a job terminates execution. This is indicated by the time t_i, for $i = 1 \cdots q$ and n_i represents the number of idle nodes between *i* and $i + 1$.

VII. Analysis of Results

A. Input Parameters

The simulations were performed using a simulation application implemented in Java, which was developed by the Research Group in Applied Computational Modeling, allowing the simulation of entities in systems of parallel and distributed computing. Therefore, this application was developed for the following purposes: analysis of the mechanisms used in dispatchers, responsibility for the distribution of jobs in the clusters, and evaluation of different scheduling algorithms covered in this work.

In this environment, we assume that two clusters (32 and 64 processors) belong to the administrative domain, so that they are able to communicate with the GD. For analysis of the environment, we use various traces extracted from a real distributed environment. However, the workload used for the experiment consists of 1,500 jobs in total and 24,152 tasks, which are described by the tuple (*id*, *at*, s_j, *pt*), Section III. Furthermore, the workload used in the simulation of job, *j*, has different characteristics, such as size of each job, processing time, among others. The mapping between tasks and processors is one to one, with a total of 96 processors in the system. For the simulation, we proposed three scenarios: i) in the S1, the schedulers AFCFS and LGFS (without migration) were used; ii) in the S2, the schedulers AFCFS and LGFS with migration mechanisms (AFCFSm) and (LGFSm) were applied; iii) and in the S3, the schedulers AFCFSm and LGFSm with the adaptation strategy on the Local Dispatcher (aLD) were used.

It is noteworthy that in all three scenarios, we applied the strategy of GD as decision making in the distribution of jobs among clusters. For each scenario, ten simulations were performed, from which we calculated the average values of waiting times, response times and LoC, with a confidence interval of 95%. In the next section, we present the results of simulations performed, using the metrics as described in Section VI.

B. Simulation Results

The results that follow describe the impact on system performance mentioned above in relation to migration in the applied schedulers group AFCFS and LGFS. Furthermore, the impact of adaptive place order is examined.

- Average response time versus number of jobs executed

In Fig. 2, the ART is illustrated in three scheduler scenarios, AFCFS and LGFS without migration (S1), AFCFSm and LGFSm with migration (S2), and AFCFSm and LGFSm with migration and aLD (S3), respectively, where the *x-axis* represents the number of jobs executed. Note that we take into account the response time increase of jobs rescheduled.

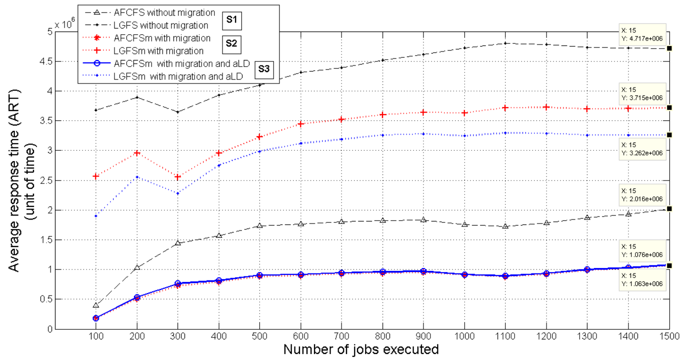

Figure 2. ART versus No. of executed Jobs - S1, S2 and S3

In the three scenarios, as illustrated in Fig. 2, it can be seen that the AFCFS had the lowest ART of all amounts of jobs performed with respect to LGFS. Furthermore, the ART showed an increase that conformed to the number of jobs carried out. This is justified for two reasons: firstly, an increase of executed jobs, *j* and *f*, and secondly, the processing time of the jobs are different. The scenario S2 shows a fairly significant reduction in the ART in relation to the scenario S1. This shows that the use of migration causes great impact on reducing the response time. Therefore, the suggested method could use the available processors more efficiently, thus reducing the fragmentation, and subsequently, the response time.

At the S3 stage (with the migration and aLD), Fig. 2, the ART was reduced in comparison to the S1 and S2 scenarios. This occurred for three reasons: firstly, the use of the migration strategy was implemented, which presented satisfactory impact on the results, as mentioned above; secondly, the migration with aLD, the JSQ algorithm distributes the tasks in the queue more fairly; and thirdly, the aLD acts before the schedulers begin to queue, thus minimizing the limitations of these at the time of allocation of tasks to resources. Furthermore, it can be seen that the scheduler LGFS in the scenario S3 ($ART = 1.076e{+}006$) showed a small reduction in the average response time with respect to S2 ($ART = 1.063e + 006$). The case scheduler AFCFS now visibly presented the best result in the three scenarios. The information concerning the total task, i.e., the number of jobs in the queue over existence or non existence of a running task on the processor, implies a reduction of task waiting time and response time. Furthermore, we observed a reduction in the number of migrations, causing direct impact on improvement of the system.

- Utilization of clusters

In this section, the performance analysis based on the use of resources using the three scenarios S1, S2 and S3 is presented. Fig. 3 illustrates the percentage utilization of the clusters in each scenario based on the number of interactions. In Fig. 3 (S1), on the intervals $400 - 2600$ (aFCFS) and $400 - 2700$ (LGFS), the average utilization of clusters is 50% and $42,5\%$, respectively. The LGFS (S1) showed an increase in the range from $1600 - 2400$. This happens because this algorithm takes care of larger jobs. Therefore, LGFS tend to offer greater fragmentation in the system.

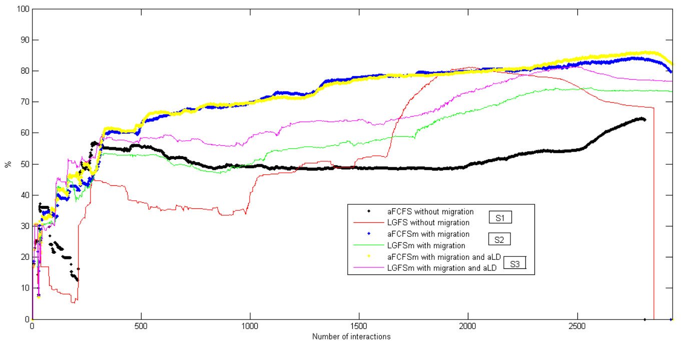

Figure 3. Utilization (%) versus number of interactions - S1, S2 and S3

In Fig. 3 AFCFSm and LGFSm (S2), the average utilization of the clusters is 60% and 50%, respectively. These results show an increase compared to the results of the scenario (S1), even with the arrival of high priority tasks in the environment. Furthermore, it can be seen that algorithms with the migration strategy could more efficiently use the resources. The results in Fig. 3 algorithms AFCFSm and LGFSm aLD (S3) show an increase of 10% over S2. That is, the average resource utilization is 75%. The scenario S3 distributes the tasks in the fairest way to the processors, causing direct impact on improving resource utilization.

- LoC versus Scenarios

Fig. 4 illustrates the loss of system capacity by fragmentation in three scenarios S1, S2 and S3. In the scenario S1, the scheduling policy AFCFS presents the lowest of LoC $(30, 4\%)$ compared to LGFS ($LoC = 31, 1\%$). This result confirms that the LGFS tends to offer greater fragmentation in the system, since this algorithm favours regarding jobs with a larger number of tasks. The scenario S2 shows a considerable decrease in fragmentation compared to S1, confirming once again that the migration minimizes fragmentation in the system. Furthermore, it can be seen that the AFCFS presents a lower percentage relative to LGFS. This implies that the AFCFS with migration is able to schedule jobs more effectively and subsequently reduces fragmentation. The S3 scenario shows the best results compared to the others. Therefore, the strategy implemented in LD offers an efficient scheduler.

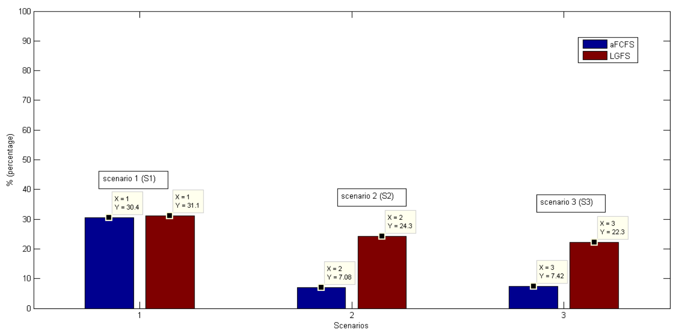

Figure 4. LoC (in %) - Fragmentation in three scenarios

VIII. Conclusion and Future Work

In this work, experiments were carried out using a heterogeneous environment based multi-cluster, a structure of two layers which were applied to different scenarios. For analysis of these experiments, we used performance metrics to evaluate the performance of schedulers in different situations.

The ART analysis results indicated a reduction of response times by implementing the migration mechanism, (Section V-A), in the schedulers (AFCFS and LGFS). This implied that the suggested method of migration could use the idle processors more efficiently and therefore reducing the fragmentation. Comparing results of the scenarios S2 and S3, we conclude that the strategy implemented in LD has a better response time. This shows that the adaptive in order site (aLD), the algorithm distributes JSQ queues of tasks more efficiently by minimizing the waiting time of the task, as well as response time. From the results of the simulations, one can observe the reduction of migration, causing a direct impact on efficiency. The algorithm AFCFS in ART metric shows the best results when compared with LGFS in the three scenarios. Regarding the utilization of metrics clusters, it was confirmed that the migration technique minimizes idle processors in the system, as well as fragmentation, with the most significant results obtained with the further scenario (S3). The latter was even more efficient, reducing the overhead on the system caused by excessive migration. The LoC metric measures the impact that the schedulers bring to the system in relation to fragmentation. The results obtained in Fig. 4, AFCFS without migration algorithm (30,4%), were achieved through less fragmentation with respect to LGFS $(31, 1\%)$. With the suggested method of migration, fragmentation was considerably reduced in AFCFS $(7, 08\%)$ and LGFS $(24, 3\%)$, and with the implementation aLD, the results were more than satisfactory. This still showed that AFCFSm caused less fragmentation with the aLD system in relation to schedulers (LGFSm with migration and LGFSm with aLD). From the results, we aim to reduce the fragmentation through the controlled use of task migration between the rows of multi-cluster processors in a heterogeneous environment, as well as better use of them, implying a reduction in operating costs by the service providers in QoS expectations of the users.

As a future perspective, we must examine the proposal in other heuristic algorithms, comparing it with them schedulers used in the different approaches of this work. Furthermore, there is a proposal to implement the proposal in a real environment.

References

[1] J. M. U. de Alencar, R. Andrade, W. Viana and B. Schulze, P2P scheme: a P2P scheduling mechanism for workflows in grid computing, Concurrency and computation: Practice and experience, John Wiley Sons, Ltd, 2011.

[2] H. Luo, D. Mu, Z. Deng and X. Wang, A review of job scheduling for grid computing, Research of computer, vol. 22, 2005, pp. 16-19.

[3] H. Topcuoglu, S. Hariri and M. S. Wu, Performance-effective and low complexity task scheduling for heterogeneous computing, IEEE Transactions parallel distributed systems, vol. 13, 2002, pp. 260-274.

[4] T. L. Casavant and J. G. Kuhl, A taxonomy of scheduling in general-purpose distributed computing systems, Transactions on software engineering, vol. 14, Feb. 1988, pp. 141-154.

[5] T. Ma, Q. Yian, W. Lu, D. Guan and S. Lee, Grid Task Scheduling: Algorithm review, IETE Technical, vol. 28, Apr. 2012, pp. 158-167.

[6] J. Ousterhout, Scheduling techniques for concurrent systems, Proc. of the 3rd ed. Intl. Conference on distributed computing systems, 1982, pp. 22-30.

[7] Z. Papazachos and H. D. Karatza, Performance evaluation of bag of gangs scheduling in a heterogeneous distributed system, Journal of systems and software, vol. 83, Jan. 2010, pp. 1346-1354.

[8] X. Wang, Z. Zhu, Z. Du and S. Li, Multi-cluster load balancing based on process migration, Lecture notes in computer science, Springer, Berlin, vol. 4847, 2007, pp. 100-110.

[9] H. D. Karatza, Performance of gang scheduling strategies in a parallel system, Simulation modeling practice and theory, Elsevier, vol. 17, Feb. 2009, pp. 430-441.

[10] Z. Papazachos and H. D. Karatza, Gang scheduling in multi-core clusters implementing migrations, Future generation computer systems, vol. 27, Feb. 2011, pp. 1153-1165.

[11] I. Moschakis and H. D. Karatza, Evaluation of gang scheduling performance and cost in a cloud computing system, The journal of supercomputing, vol. 59, Feb. 2012, pp. 975-992.

[12] Z. Papazachos and H. D. Karatza, Gang Scheduling in a two-cluster system implementing migrations and periodic feedback, Transactions of the society of modeling and simulation international, vol. 87, Dec. 2011, pp. 1021-1031.

[13] S. K. Garg, S. Venugopal, J. Broberg and R. Buyya, Double auction-inspired meta-scheduling of parallel applications on global grids, Journal parallel distrib. Comput, vol. 73, Apr. 2013, pp. 450-464.

[14] Z. K. Gkoutioudi and H. D. Karatza, Multi-criteria job scheduling in grid using an accelerated genetic algorithm, Journal grid computing, vol. 10, Mar. 2012, pp. 311-323.

[15] V. J. Leung, G. Sabin and P. Sadayappan, Parallel job scheduling policies to improve fairness: A case study, ICPP Workshops, 2010, pp. 346-353.

[16] W. Tang, Z. Lan, N. Desai, D. Buettner and Y. Yu, Reducing fragmentation on torus-connected supercomputers, Parallel & distributed processing symposium, 2011, pp. 107-115.

[17] W. Tang, D. Ren, Z. Lan and N. Desai, Adaptive metric-aware job scheduling for production supercomputers, 41st International conference on parallel processing workshops, 2012, pp. 107-115.

[18] D. Feitelson (2014, Mar.) The Standard Workload Format. [Online]. Available:http://www.cs.huji.ac.il/labs/parallel/workload/swf.html

[19] Y. Zhang, H. Franke, J. Moreira and A. Sivasubramaniam, The impact of migration on parallel job scheduling for distributed systems, Proceedings of europar, 2000, pp. 242-251.

[20] A. Burkimsher, I. Bate and L. S. Indrusiak, 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, vol. 29, Oct. 2012, pp. 2009-2025.