Performance Evaluation of Gang Scheduling in a Two-Cluster System with Migrations

Zafeirios C. Papazachos and Helen D. Karatza
Department of Informatics
Aristotle University of Thessaloniki
54124 Thessaloniki, Greece
{zpapazac, karatza}@csd.auth.gr

Abstract

Gang scheduling is considered to be a highly effective task scheduling policy for distributed systems. In this paper we present a migration scheme which reduces the fragmentation in the schedule caused by gang scheduled jobs which cannot start. Furthermore, the existence of high priority jobs in the workload is addressed by the proposed strategy. High priority jobs need to be started immediately, which can in turn lead to the interruption of a parallel job's execution. A distributed system consisting of two homogeneous clusters is simulated to evaluate the performance. Our simulation results indicate that the proposed strategy can result in a performance boost.

1. Introduction

Distributed systems consisting of multiple clusters have emerged as a solution to the increasing demand in computing resources. The scheduling strategy that might be used in such a system is of great significance since the performance achieved is proportional to the algorithm's effectiveness.

In a distributed system the scheduling algorithm is responsible of allocating processors to the existing jobs. Particularly in the case of parallel jobs which consist of frequently communicating tasks, a scheduling algorithm which allows the tasks to execute concurrently would be suitable. Gang scheduling is an effective approach of scheduling such jobs. Gang scheduling relies on time-space sharing. The main concept of this technique is to group the tasks of a parallel job, thus formulating a gang, and execute them simultaneously on different processors. In this way, we eliminate the risk of waiting on a task which is currently not running on any processor. It is obvious from the above that the number of tasks in a gang cannot exceed the number of available processors.

In order to avoid fragmentation, the tasks of a parallel job could be dynamically migrated hence delivering the desired flexibility of adjusting the schedule. Although migration enables dynamic load distribution ([3], [12], [13], [16], [17]), it has not achieved widespread use due to its complex nature. In a multi-cluster environment the use of migration for load balancing purposes is more obvious, since the current workload of each cluster might differ in time.

Another major issue regarding distributed systems is the workload composition. There is a great impact on system performance when high priority jobs exist in the workload. It might be necessary for a high priority job to interrupt a parallel job's execution in order to satisfy its immediate requirements. In this way, the processing of a gang is delayed so as to maintain a certain Quality-of-Service (QoS) level in favor of a high priority job.

Gang scheduling is the case of scheduling parallel jobs where their tasks need to very frequently communicate with each other. Therefore, any method that involves context switching between different tasks can cause high overhead due to the fact that the current network traffic between tasks of the same gang should be saved and restored at a later time. Furthermore, some messages that should be received by a process before being switched may be received by another process after being switched. Considering this, it is impractical to migrate or preempt a running task which belongs to a gang. Especially in the event of a high priority job arrival, preempting instead of interrupting a gang will stall the execution of the high priority job. However, in this paper migration involves the movement of a task from a local queue to the head of another queue, rather than the movement of a running application. In this way, high overheads that may in other case occur are avoided.

978-1-4244-3750-4/09/$25.00 ©2009 IEEE

There are a great many studies on gang scheduling policies ([4], [6-11], [18]). Karatza in [11] studies the performance of the Adapted First Come First Served (AFCFS) and the Largest Job First Served gang scheduling policies in the presence of critical sporadic non-parallel jobs. In [18] a combination of gang scheduling and migration is examined. However, those gang scheduling policies previously studied do not consider implementing both high priority jobs in the workload and migrations at the same time. They all differ in the way the resources are shared and a multi-cluster environment is not always considered.

In this paper, a simulation model consisting of two homogeneous clusters is considered. The workload consists of parallel jobs (gangs) and high priority non-parallel jobs which can overtake gangs. We compare the performance of the AFCFS algorithm with a modified version which implements migrations under various workloads. Furthermore, we make use of reservation and aging techniques in order to regulate the number of migrations occurring and avoid excessive overhead.

The structure of this paper is as follows. Section 2 gives a description of the system and the workload models. Section 3 describes the scheduling policies and Section 4 presents the metrics used to assess the performance and analyzes the simulation results. Finally, Section 5 provides some concluding remarks and suggestions for future research.

2. System and workload models

A simulation model is used, instead of measurements of real systems, in order to evaluate the system's performance. Simulation models enable us to study desired algorithms in detail. The suggested scheduling strategy can be implemented in a real system. Workload traces within a grid would also be useful for the performance evaluation. However, such traces currently do not exist. Therefore, workload models are commonly used to evaluate an algorithm's performance [5].

The simulation model consists of two clusters. Each cluster is a homogeneous distributed system consisting of P=16 processors, each serving its own queue (Figure 1). We assume that the processors in each cluster are interconnected via a high speed local area network. The clusters are connected to each other via a wide area network.

In this paper we assume that the communication between the processors is contention-free. Thus, we consider that the communication latencies are included implicitly in the gangs' execution time. However, overheads, which occur when migration schemes are applied, are considered.

The service time of a job is exponentially distributed with mean of $1/\mu$. The workload consists of parallel jobs (gangs) and non-parallel high priority jobs.

Gangs require a number of processors equal to its number of tasks for their execution. The number of job tasks is uniformly distributed in the range of [1..P]. Thus, the mean number of tasks per gang is equal to (1+P)/2. The mean inter-arrival time of gangs is exponentially distributed with a mean of $1/\lambda_1$.

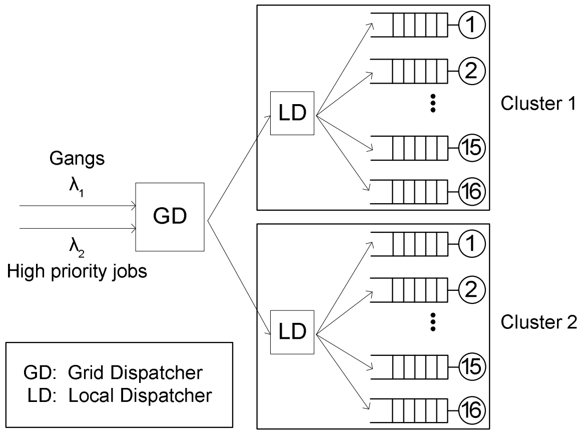

Figure 1. The queuing network model

In this study we consider high priority jobs which consist of a single task. These jobs need to start their execution immediately after their arrival. As a result of this, an occupied processor by a task previously arrived, might have to be interrupted in order to execute a high priority task. If a task is interrupted, all the progress made is lost and all the work must be redone. Furthermore, since a high priority job cannot be interrupted, that leaves us only with the choice of a task belonging to a gang. This means that every sibling task of the associated gang has to stop its progress and be rescheduled. The mean inter-arrival time of high priority jobs is exponentially distributed with a mean of $1/\lambda_2$.

Although the suggested set up could be considered as only a part of a larger grid, any remark made in this study can be useful and applied on a larger scale grid.

3. Scheduling strategy

3.1. Job routing

The first thing that occurs when a job arrives at the distributed system is to be dispatched to one of the available clusters. This is accomplished by a *grid dispatcher.* The grid dispatcher that we employ in our system assigns jobs to the clusters in a uniformly distributed manner. The probability that the job is assigned to one of the clusters is equal. That is 50% probability in a system consisting of two clusters. In this system we consider that the grid dispatcher does not have a priori knowledge about the clusters' current load. In this way, any overhead which could result from the implementation of a cluster information feedback algorithm is avoided. Therefore, a probabilistic algorithm is considered suitable for this purpose.

After a parallel job has been send to a cluster, the *local dispatcher* assigns its tasks to the available queues. The tasks are distributed to the queues based on the "Join the Shortest Queue" (JSQ) policy. Sibling tasks (task which belong to the same gang) cannot be assigned to the same queue, since in gang scheduling there is a one-to-one mapping between tasks and processors.

When a high priority job arrives at a cluster, it is routed based on the "Join the Shortest Queue" policy. The shortest queue of the cluster is selected for the job to join it, provided that there is not already another high priority job occupying the aforementioned processor. In this study, λ_2 values are quite small (λ_2 = 1/5, 1/10) compared to the total number of available processors in the system (32 processors, where each processor can serve on average 1 high priority job per time unit). Therefore, this system can on average serve 32 high priority jobs per time unit. Therefore, by using these small λ_2 values we do not encounter any situations where a high priority job can find all processors busy serving high priority jobs. However, even a small arrival rate λ_2 of high priority jobs can have a serious impact on the performance of gangs. This is because when a high priority job arrives at a processor which serves a gang task, then not only the running task but also all the sibling tasks of the associated gang will be interrupted. Moreover, it is not certain whether the remaining processors which were forced to stop the gang execution are going to be scheduled immediately for another gang execution. This is because either no gang that matches the scheduling criteria may be found or the interrupted gang had previously migrated resulting in reservation of its assigned processors (as explained in section 3.3).

3.2. Gang scheduling

The Adapted-First-Come-First-Served (AFCFS) algorithm is applied to each cluster separately. According to this method, a job is scheduled whenever processors assigned to its tasks are available. When a job whose tasks are waiting in front of the queues cannot start its execution, AFCFS policy schedules other jobs whose tasks are behind the tasks of the aforementioned job.

AFCFS tends to favor jobs which consist of a small number of tasks and therefore require a smaller number of processors. However, this practice results in an increase of the large jobs' response time.

If the processing of a gang, which was scheduled using the AFCFS method, is interrupted by a high priority job, then all its tasks are rescheduled for execution at the head of their associated queues. Furthermore, any progress made for the specific gang is lost and must be redone.

3.3. Migration

A common problem that occurs when using gang scheduling is that there is a number of processors which remain idle even though there are tasks in their queues. This is due to the fact that the execution of a gang does not start unless all of its tasks can start their execution simultaneously. Therefore, the main reason for the delay of gang execution is that one or more of its tasks are waiting in queues which belong to busy or reserved processors. Implementing migrations is a way to avoid this kind of fragmentation. Migration involves the transferring of a task from one remote queue to the head of a local queue. Although migration seems to be an ideal solution to the fragmentation issue, it should be exercised with caution since migration causes an overhead. The irrational use of migration could lead to higher response time and other undesirable results.

In our simulation model, we examine the impact of migrating tasks both locally and through the grid. A local migration is considered to be the transferring of a task from one processor queue to another queue which belongs to the same cluster. A grid migration is more complicate because it involves the transferring of one task from one cluster to another. A major difference between local and grid migration is the higher overhead of the latter.

The local migration algorithm works as follows. Although there are available processors in a cluster, the AFCFS algorithm might fail to schedule a gang due to the fact that there is not a one-to-one mapping between its tasks and processors. Whenever this kind of fragmentation occurs, we examine which gangs have at least one waiting task at the head of an available processor's queue. From these gangs we select the one that requires the least number of migrations to start its processing. Any parallel jobs that their number of tasks exceeds the number of available processors are excluded from this procedure. The migrated tasks are transferred to the head of the queues. Consequently, the job can start its execution immediately once the migration is completed.

If any available processors still remain, we examine which gangs have a task waiting on the head of their queue. Provided that there are enough available processors in the other cluster, we select the gangs which require the smallest number of migrations to start their execution. During a task migration, the target node is reserved in order to prevent other tasks from seizing it. By reserving the target processor we ensure that after the migration has finished, the gang will have an available processor for all of its tasks and its execution will start immediately. The only possible cause that might hinder its execution is the arrival of a high priority job.

In the event of a high priority job arriving at a queue, the parallel job that is occupying the corresponding processor is interrupted in order to give its place to the high priority job. Unlike the gangs scheduled using the AFCFS algorithm, any parallel jobs with migrated tasks interrupted by a high priority job are reserved. This means that the processors previously occupied by the gang are only allowed to serve high priority jobs. In this way, when there are not any high priority jobs occupying the necessary processors, the reserved gang is brought back to execution immediately without having to wait other gangs to finish their execution first. Consequently, parallel jobs are allowed to migrate only once in their makespan. The reservations method saves the system from excessive network traffic due to migrations, does not let the parallel jobs to starve and keeps the response time of the migrated gangs low. Furthermore, the reservations technique allows the gangs which have migrated tasks in a remote cluster to restart their execution after being interrupted. Otherwise, those gangs would not be able to restart their processing, because AFCFS algorithm is not suitable to schedule parallel jobs which have tasks in more than one cluster.

We also make use of aging, in order to regulate the number of migrations which occur in the system. A queue is not available for other tasks to migrate when there are tasks in the queue which have already granted priority three times in the past. In this way, the starvation of the tasks belonging to this queue is prevented.

The scheduling hierarchy that is implied should be noted. The main scheduling algorithm is the AFCFS. Then the local migrations method attempts to schedule the gangs which are not possible to be scheduled using the AFCFS algorithm. Eventually, grid migrations are used in an attempt to utilize more resources by scheduling a few more gangs. The underlying reason of this hierarchy is the overhead that is imposed by each of these steps. Unlike migration techniques, AFCFS does not cause any additional overhead. For this reason it is desirable to schedule most of the jobs using the AFCFS algorithm. The grid migration overhead is significantly higher than that of local migrations. That is why local migrations precede of the grid migrations.

4. Performance evaluation

4.1. Performance metrics

In order to evaluate the system's performance we employ the following metrics.

Response time r_j of a parallel job j is the time interval from the arrival of the job to the grid dispatcher to the service completion of this job. Note that since the grid dispatcher does not use a queue to route the jobs, we consider that there is no waiting on the grid dispatcher.

Slowdown s_j of a gang j is the response time of this job divided by the job's service time. This metric is used to measure the delay of a job against its actual runtime. If e_j is the execution time of the job j, then the slowdown is defined as follows:

$$s_j = r_j / e_j$$

Let m be the number of the total processed parallel jobs. The following metrics used for performance are defined as follows ([11], [15]):

- The average response time RT:

$$RT = \frac{1}{m} \times \sum_{j=1}^{m} r_j$$

- The average slowdown SLD:

$$SLD = \frac{1}{m} \times \sum_{j=1}^{m} s_j$$

Additionally, the response time and the slowdown of each job are weighted [15] with its size. In this way, it is avoided that jobs with the same execution time, but

Table 1. Notations

P	number of processors in a cluster
μ_1	mean service rate of gangs
$1/\mu_1$	mean service demand of gangs
μ_2	mean service rate of high priority jobs
$1/\mu_2$	mean service demand of high priority jobs
λ_1	mean arrival rate of gangs
λ_2	mean arrival rate of high priority jobs
U	average processor utilization
RT	average response time of gangs
D_{RT}	Relative (%) decrease in RT when migrations are implemented
WRT	Average weighted response time
D_{WRT}	Relative (%) decrease in WRT when migrations are implemented
SLD	Average slowdown
D_{SLD}	Relative (%) decrease in SLD when migrations are implemented
$WSLD$	Average weighted slowdown
D_{WSLD}	Relative (%) decrease in $WSLD$ when migrations are implemented
k	Maximum number of times that a task can be bypassed by other tasks which migrate to the same queue

with different number of parallel tasks, have the same impact on the overall performance. Let *p(x)* represent the number of processors required by job x. The following weighted metrics are used:

- The average weighted response time *WRT*:

$$WRT = \frac{\sum_{j=1}^{m} p(x_j) \times r_j}{\sum_{j=1}^{m} p(x_j)}$$

- The average slowdown *SLD*:

$$WSLD = \frac{\sum_{j=1}^{m} p(x_j) \times s_j}{\sum_{j=1}^{m} p(x_j)}$$

Table 1 sums up the parameters used in simulation computations.

4.2. Input parameters

The aforementioned queuing network model is implemented with discrete event simulation [1] using the independent replications method. Each result presented is the average value that is derived from 10 simulation experiments with different seeds of random numbers. Each simulation run is terminated upon the successful completion of 64000 gangs. We considered this simulation length long enough to derive results as we found by experimentation that longer runs did not affect the performance results significantly. The use of sufficiently long simulation runs is one of the ways that the effect of initial bias on simulation output can be reduced [2].

A parallel job requires 8.5 processors on average ((P+1)/2). There is a total of 32 processors in the system considering that there are two clusters which consist of P = 16 processors. The mean service demand of jobs is:

$$1/\mu_1 = 1/\mu_2 = 1.$$

This means an average of 32/8.5 = 3.7647 parallel jobs can be served each unit of time, assuming there are no high priority jobs and all processor are busy. So, a λ_1<3.7647 should be chosen. Arrivals of high priority jobs do appear nevertheless. That said, the available number of processors to gang service becomes $[32-(\lambda_2/\mu_2)]$. Thus, λ_1 should satisfy the following condition in order to maintain the system stability:

$$\lambda_1 < \frac{32 - (\lambda_2/\mu_2)}{8.5}$$

We examine the impact of migrations in the system using the following gang arrival rates λ_1:

$$\lambda_1 = 2.4, 2.45, 2.5, 2.55, 2.6.$$

Furthermore, we study the following two cases of high priority jobs mean inter-arrival time:

$1/\lambda_2 = 5, 10.$

The overhead for a migration between nodes which belong to the same cluster is set to 0.05 time units, whereas a migration to a node of a remote cluster is set to an overhead of 0.1 time units. Note that the grid migration overhead is set really high, but even with an overhead of this size the results reveal the beneficial role of migrations in the described system. This also compensates for any additional communication overhead occurring between sibling tasks that are positioned in separate clusters.

Finally, we have set the aging of tasks (i.e. the maximum times that is possible for a task waiting in a queue to grant priority to other tasks which migrate to this queue) to $k = 3$ times maximum.

4.3. Simulation experiments

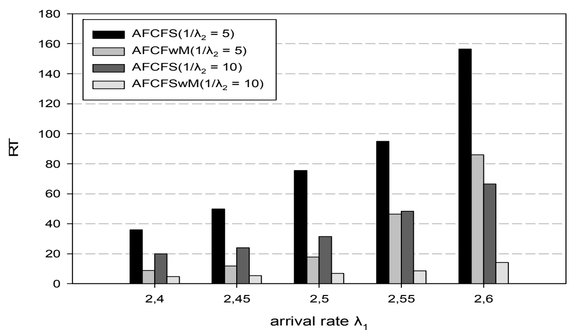

Figure 2. *RT* versus λ_1

Figure 2 depicts the response time of the AFCFS algorithm without utilizing any migrations compared to AFCFS with migrations (AFCFSwM). The use of migrations, as previously described, yields lower response time both when $1/\lambda_2 = 5$ and $1/\lambda_2 = 10$. This happens because the suggested migrations method manages to utilize the idle processors efficiently. Otherwise, due to the gangs' nature it would not be possible to serve any parallel jobs waiting on a busy processor. When a task which belongs to a gang is waiting on a busy processor, then it is possible to be transferred in order for the gang to start its execution.

Figure 3 illustrates the relative (%) decrease in RT when migrations are implemented. It should be noted that for higher workloads (both gang load $\lambda_1 = 2.55$, 2.6 and high priority job load $1/\lambda_2 = 5$) the difference in performance is decreased. There are two main reasons which cause this. First of all, the increased high priority job load causes more reservations to occur in the system. Although the reservations prevent gangs from starvation, an increased number of them prohibits other jobs from utilizing the otherwise idle processors. Another reason is that the increased workload does not leave enough available processors. If there are not enough available processors it is not easy for large gangs to migrate. Furthermore, according to the proposed migrations implementation, only the gangs that have a task at the head of an available processor's queue are examined for migration. That means there are not many candidate jobs to migrate if there are not enough available processors. However, this is done for a good reason. Migrating only the jobs that have at least one task at the head of the queue on an available processor allows for lower system complexity, less starvation for the gangs that arrived earlier and less network traffic (because not all tasks of the job will have to migrate).

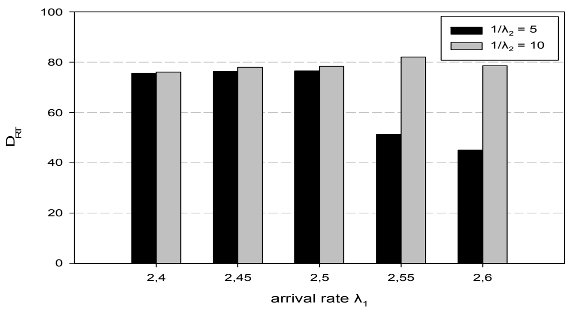

Figure 3. D_{RT} vs λ_1

Figure 4 presents the average weighted response time. Although the average response time is relatively higher when it is weighted by job degree parallelism, the results follow the same pattern in both cases. Moreover, D_{WRT} (Figure 5) is almost identical to D_{RT}. Therefore, any observations holding for RT, also apply to WRT.

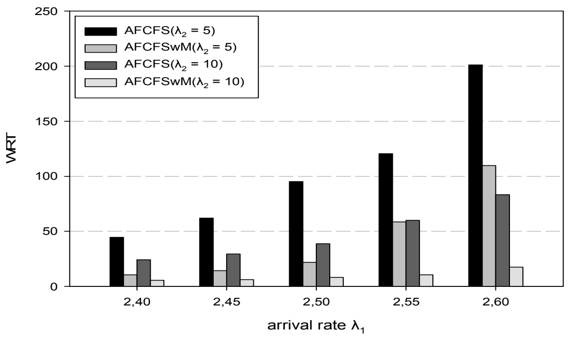

Figure 4. *WRT* versus λ_1

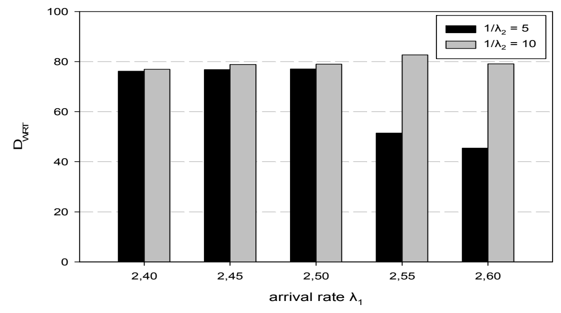

Figure 5. D_{WRT} versus λ_1

Figures 6 and 7 make it more apparent that the increased high priority jobs workload ($1/\lambda_2 = 5$) influences the impact that migrations have on the system. It is obvious that migrations yield better performance when the load of high priority jobs is kept low.

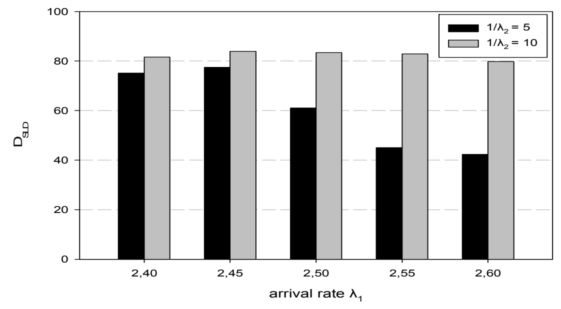

Figure 6. D_{SLD} versus λ_1

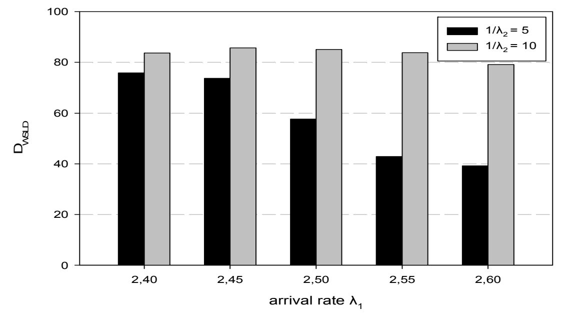

Figure 7. D_{WSLD} versus λ_1

Tables 2 and 3 show mean processor utilization in $1/\lambda_2 = 5$ and $1/\lambda_2 = 10$ respectively. In the first case utilization is higher, but this is reasonable because of the higher workload and because more high priority job arrivals cause more gangs to restart their execution.

Table 2. Mean processor utilization, $1/\lambda_2 = 5$

λ_1	AFCFS	AFCFSwM
2.40	0.6900	0.6814
2.45	0.7040	0.6930
2.50	0.7193	0.7084
2.55	0.7267	0.7281
2.60	0.7423	0.7420

Table 3. Mean processor utilization, $1/\lambda_2 = 10$

λ_1	AFCFS	AFCFSwM
2.40	0.6655	0.6593
2.45	0.6755	0.6697
2.50	0.6888	0.6842
2.55	0.7031	0.6975
2.60	0.7156	0.7123

Assigning a high priority job to a queue, which belongs to a busy processor, results in an interruption to its task execution. This in turn forces the gang to which it belongs to stop its execution. All the gang work must be repeated. So, the overall utilization appears slightly increased, because part of the processor utilization is comprised of repeated gang work due to high priority job arrivals.

5. Conclusion and future work

In this paper, we examined the impact of migrations on the performance of gang scheduling strategy in a two-clustered system in the presence of high priority jobs in the workload. Experiments were conducted using a simulation model under various workloads.

The results indicate a performance boost due to implementing the suggested migration algorithm. Even in the case of increased high priority jobs workload ($1/\lambda_2 = 5$), implementing migrations achieved better performance. Specifically, it achieved 75% decrease in response time when the gang arrival rate was low ($\lambda_1 = 2.4$) and 45% decrease in response time when gang arrival rate was high ($\lambda_1 = 2.6$)), while the arrival rate of high priority jobs was $\lambda_2 = 1/5$.

In this paper, we used a two-cluster system in order to determine whether the overhead caused by migrations is tolerable enough to achieve better performance. The study of gang scheduling with migrations in a heterogeneous system consisting of multiple clusters would be a logical extension in the future. As part of our future work, we also intend to examine more scheduling algorithms and implement additional methods of assigning jobs to clusters.

References

[1] A.M. Law, W.D. Kelton, *Simulation Modelling and Analysis*, McGraw-Hill, New York, USA, 1991.

[2] F. Neelamkavil, *Computer Simulation and Modelling*, J. Wiley and Sons, Chichester, 1987.

[3] D.Eager, E. Lazowska, and J. Zahorjan, "The Limited Performance Benefits of Migrating Active Processes for Load Sharing", *Conference on Measurement & Modeling of Computer Systems,* ACM SIGMETRICS , May 1988, pp. 63-72.

[4] D.G.,Feitelson, and M.A. Jette, "Improved Utilization and Responsiveness with Gang Scheduling", In *Job Scheduling Strategies for Parallel Processing*, *Lecture notes in Computer Science*, Springer-Verlang, Berlin, Germany, 1997, Vol. 1291, pp. 238-261.

[5] A. Iosup, D.H.J. Epema, C Franke, A. Papaspyrou, A. Schley, L. Song, R. Yahyapour, "On Grid Performance Evaluation Using Synthetic Workloads", *Lecture Notes in Computer Science,* Springer, Berlin, 2007, Vol. 4376, pp. 232-256.

[6] H.D. Karatza, "A Simulation-Based Performance Analysis of Gang Scheduling in a Distributed System", In *Proceedings of the 32nd Annual Simulation Symposium*, (San Diego, CA, April 11-15), IEEE Computer Society, Los Alamos, 1999, pp. 26-33.

[7] H.D. Karatza, "Gang Scheduling in a Distributed System with Processor Failures", In *Proceeding of the UK Performance Engineering Workshop*, University of Bristol, UK, 1999, pp. 199-208.

[8] H.D. Karatza, "Gang Scheduling Performance on a Cluster of Non-Dedicated Workstations", In *Proceedings of the 35th Annual Simulation Symposium*, IEEE Society, Los Alamitos, April 2002, pp. 115-121.

[9] H.D. Karatza, "Gang Scheduling in a Distributed System under Processor Failures and Time-varying Gang Size", In *Proceedings of the 9th IEEE Workshop on Future Trends of Distributed Computing Systems*, (San Juan, Puerto Rico, May 28-30), IEEE Computer Systems, Los Alamitos, CA, 2003, pp. 330-336.

[10] H.D. Karatza, "Scheduling Gangs in a Distributed System", *International Journal of Simulation: Systems*, Science Technology, UK Simulation Society, Vol.7 no. 1, January 2006, pp. 15-22.

[11] H.D. Karatza, "Performance of Gang Scheduling Policies in the Presence of Critical Sporadic Jobs in Distributed Systems", In *Proceedings of the 2007 International Symposium on Performance Evaluation of Computer and Telecommunication Systems*, SPECTS'07, (San Diego, CA, July 16-18), SCS, San Diego, CA, 2007, pp. 547-554.

[12] M.J. Litzkow, M. Livny, and M.W. Muttka, "Condor – A Hunter of Idle Workstations", In *Proceedings of 18th International Conference on Distributed Computer Systems*, IEEE, June 1988, pp. 104-11.

[13] D.S. Milojiĉić, F. Douglis, Y. Paindaveine, R. Wheeler, S. Zhou, "Process Migration", *ACM Computing Surveys (CSUR)*, ACM, New York, September 2000, Vol. 32, Issue 3, pp. 241-299.

[14] S. Petri and H. Langendörfer, "Load Balancing and Fault Tolerance in Workstation Clusters—Migrating Groups of Communicating Processes", *Operating Systems Rev*., Oct. 1995, vol. 29, Vol. 4, pp. 25-36.

[15] A. Streit, "Enhancements to the Decision process of the Self-Tuning dynP Scheduler", *Lecture Notes in Computer Science*, Springer, Berlin, May 2005, Vol. 3277, pp. 63-80.

[16] X.Y. Wang, Z.Y. Zhu, Z.H. Du and S.L. Li, "Multi-Cluster Load Balancing Based on Process Migration", *Lecture Notes in Computer Science*, Springer, Berlin, 2007, Vol. 4847, pp. 100-110.

[17] Y. Zhang, H. Franke, J. Moreira, and A. Sivasubramaniam, "The Impact of Migration on Parallel Job Scheduling for Distributed Systems", In *proceedings of Europar*, (Munich, Germany, 29 August to 2 September), Europar, 2000, pp. 242-251.

[18] Y. Zhang, H. Franke, J. Moreira, and A. Sivasubramaniam, "An Integrated Approach to Parallel Scheduling Using Gang-Scheduling, Backfilling an Migration", *IEEE Transactions on Parallel and Distributed Systems*, IEEE Computer Society, Los Alamitos, CA, 2003, Vol. 14, no. 3, pp. 236-247.

Biography

Zafeirios Papazachos is a PhD student at the Department of Informatics of the Aristotle University of Thessaloniki, Greece. His research interests include: modeling, simulation and performance evaluation of scheduling algorithms in grid systems.

Helen Karatza is an Associate Professor at the Department of Informatics of the Aristotle University of Thessaloniki, Greece. Her research interests include: performance evaluation of parallel and distributed systems, scheduling and simulation.