Scheduling Gangs with Different Distributions in Gangs' Degree of Parallelism in a Multi-Site System

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. Gangs are jobs which consist of a number of interacting tasks which are scheduled to run simultaneously on distinct processors. Simulation experiments are conducted to address performance issues which concern the impact of the variability in gangs' degree of parallelism in the case of implemented migrations. A simulation model consisting of two sites is used to provide results on the performance of the system.

1. Introduction

In a distributed system, the scheduling algorithm is an important aspect of the overall system performance. Gang scheduling is considered to be an effective approach of scheduling parallel tasks which need to frequently communicate with each other. According to this technique [2], the tasks of a parallel job are grouped together and executed simultaneously on different processors. In this way gang tasks interact efficiently by busy waiting, without the risk of waiting on a task which is currently not running on any processor. 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 [13]. Although migration enables dynamic load distribution ([1], [3], [7], [12]), it has not achieved widespread use due to its complex nature. In a multi-site environment the use of migration for load balancing purposes is more obvious, since the current workload of each site might differ in time. 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.

The paper is organized 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. Section 5 presents the related work. Finally, Section 6 provides some concluding remarks and suggestions for future research.

2. System and workload models

The simulation model consists of two sites. Each site is a homogeneous distributed cluster consisting of $P = 16$ processors, each serving its own queue (Figure 1). In this study the terms cluster and site are used interchangeably.

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).

Gangs require a number of processors equal to their number of tasks for their execution. We examine the following two cases with regard to the distribution of the number of the gang tasks:

- In the first case we assume a "bounded" normal distribution for the number of gang tasks in the range of $[1..P]$ with mean $\eta = (1+P)/2 = 8.5$. We have studied the following two cases for the standard deviation: $\sigma = 3$ and $\sigma = 4$.
- In the second case, the number of gang tasks is uniformly distributed in the range of $[1..P]$. Thus,

the mean number of tasks per gang is equal to $\eta = (1+P)/2 = 8.5$.

The mean inter-arrival time of gangs is exponentially distributed with a mean of $1/\lambda$.

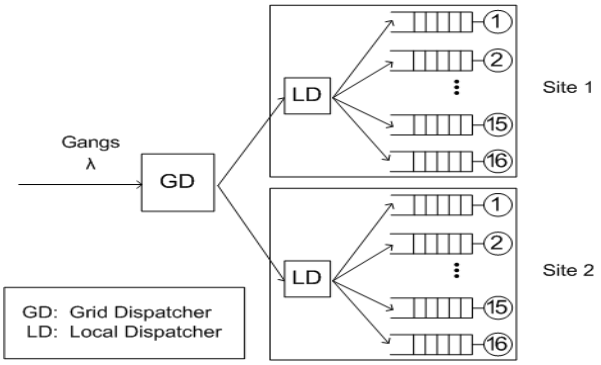

Fig. 1. The queuing network model

3. Scheduling strategy

3.1. Job routing

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

After a parallel job has been send to a site, 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 (tasks 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.

3.2. Gang scheduling

The Adapted-First-Come-First-Served (AFCFS) algorithm is applied to each site 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.

3.3. Migration

Gangs require all of their tasks to 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 queue to the head of another queue [14]. 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 large 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 site. A grid migration is more complicate because it involves the transferring of a task from one site 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.

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.

4. Performance evaluation

4.1. Performance 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.

Slowdown s_j of a gang j is the response time of this job divided by the job's service time. If e_j is the execution time of the job j, then the slowdown is defined as $s_j = r_j / e_j$.

Table 1 sums up the simulation parameters:

Table 1. Notations

P	Number of processors in a site
μ	Mean service rate of gangs
$1/\mu$	Mean service demand of gangs
λ	Mean arrival rate of gangs
σ	Standard deviation
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

Additionally, the response time and the slowdown of each job are weighted with its size ([4], [11]). Let $p(x)$ represent the number of processors required by job x. The number of the total processed parallel jobs is represented by m. 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:

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

4.2. Input parameters

The aforementioned queuing network model is implemented with discrete event simulation [6] 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 [8].

A parallel job requires 8.5 processors on average. There is a total of 32 processors in the system considering that there are two sites which consist of P = 16 processors.

The mean service demand of jobs is $1/\mu = 1$. This means that an average of 32/8.5 = 3.7647 parallel jobs can be served each unit of time, assuming that all processors are busy due to gang service. So, a λ<3.7647 should be chosen. However, due to gang scheduling there are often idle processors although there jobs in the queues. Therefore, the queues get easily saturated when λ is close to 3.7647.

We examine the system performance for the following gang arrival rates: λ = 2.3, 2.35, 2.4, 2.45.

The overhead for a migration between nodes which belong to the same site is set to 0.05 time units, whereas a migration to a node of a remote site is set to an overhead of 0.1 time units. 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 results and discussion

Figures 2a-b depict the response time for gangs whose number of tasks is normally distributed and uniformly distributed respectively. In both Figures 2a-b AFCFS with migrations (AFCFSwM) yields lower mean response time. For $\sigma = 3$ (Figure 2a) AFCFS has a larger response time than in the $\sigma = 4$ case. This is because in the $\sigma = 3$ case there are more gangs whose size is close to the mean value of 8.5. In this case it is very difficult to find gangs in the queues of which all their tasks have one-to-one mapping to an idle processor. Gangs with a small number of tasks are more probable to exactly match the remaining idle processors. The number of gangs with small size is smaller for $\sigma = 3$ than for $\sigma = 4$. The largest number of gangs with a small size appears in the case of uniformly distributed gang size (Figure 2b). Therefore, in Figure 2b AFCFS presents lower mean response time than in Figure 2a.

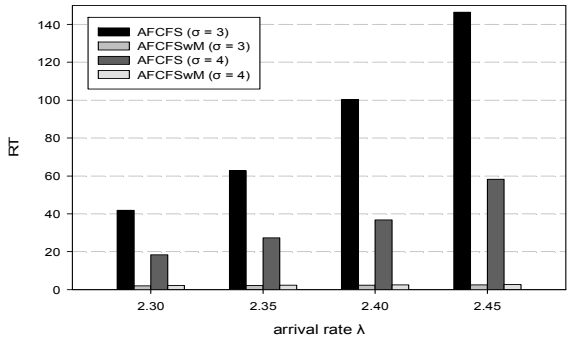

Fig. 2a. *RT* vs λ, truncated normal distribution

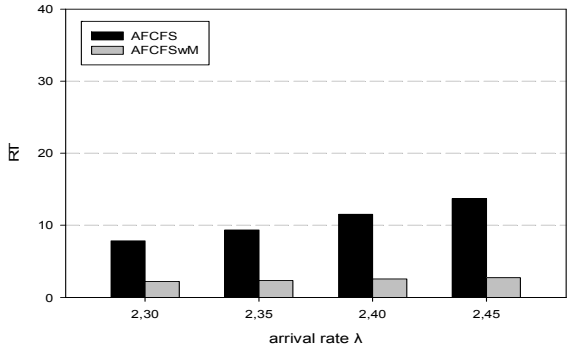

Fig. 2b. *RT* vs λ, uniform distribution

It is also apparent that AFCFSwM is not affected by the variability of the gang size as much as AFCFS does. This is because the suggested migrations scheme manages to utilize the idle processors efficiently. Otherwise, due to the gangs' nature it would not be possible to serve any parallel jobs in the case where one of their tasks waits on a busy processor. When using migrations, it is possible for a gang task to be transferred in order for the gang to start its execution. Furthermore, by using grid migrations it is possible to utilize idle processors from both sites in order to serve a gang which is otherwise too large to be served by the remaining idle processors of a single site. However, this causes an overhead which affects the response times of the gangs.

Figures 3a-b present the relative % decrease in mean response time when the migration scheme is implemented for gangs with normally distributed and uniformly distributed gang size respectively. In both cases D_{RT} generally increases with increasing load. Furthermore, D_{RT} is higher for $\sigma = 3$ than for $\sigma = 4$ (Figure 3a).

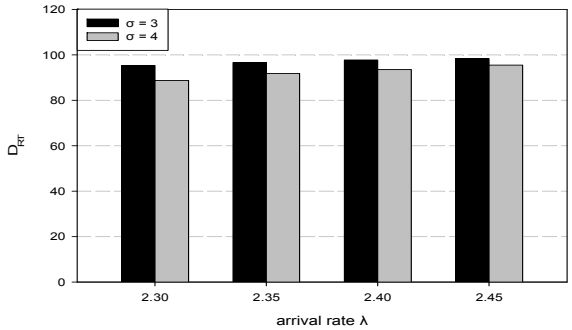

Fig. 3a. D_{RT} vs λ, truncated normal distribution

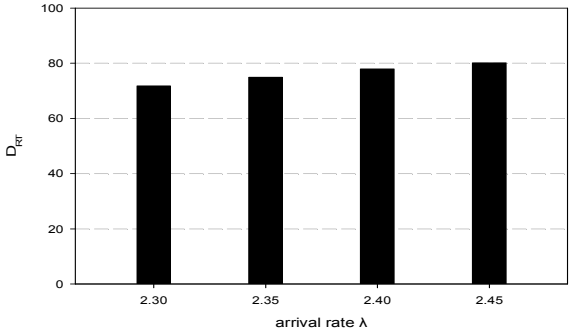

Fig. 3b. D_{RT} vs λ, uniform distribution

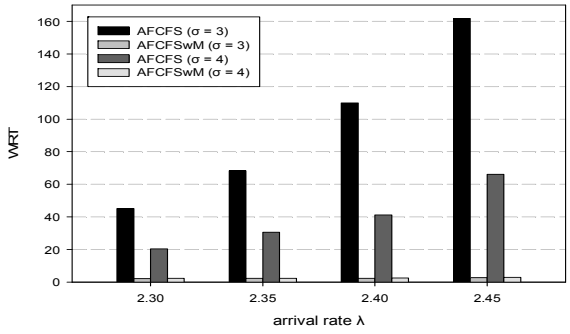

Fig. 4a. *WRT* vs λ, truncated normal distribution

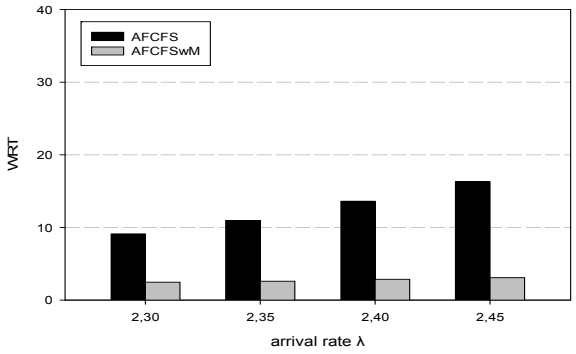

Fig. 4b. *WRT* vs *λ*, uniform distribution

In Figures 4a-b WRT appears to be higher than RT. D_{WRT} values (Figures 5a-b) follow a similar pattern to D_{RT} values.

Figures 5a-b present the relative (%) decrease in *SLD* when migrations are implemented for gangs which their degree of parallelism varies according to normal or uniform distribution respectively. When the gang size is normally distributed (Figure 5a) we observe that D_{SLD} is larger than in the case of gangs which have a uniformly distributed size (Figure 5b). In Figures 6a-b we observe that D_{WSLD} follows the same pattern as D_{SLD}.

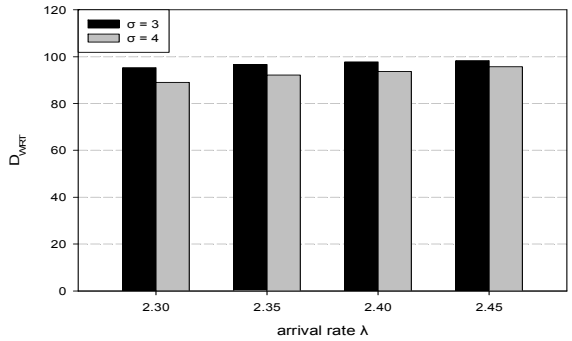

Fig. 5a. D_{WRT} vs *λ*, truncated normal distribution

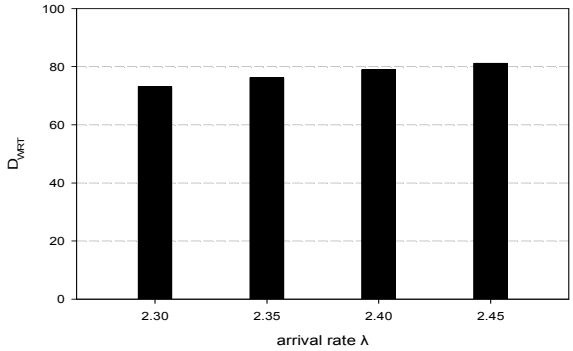

Fig. 5b. D_{WRT} vs *λ*, uniform distribution

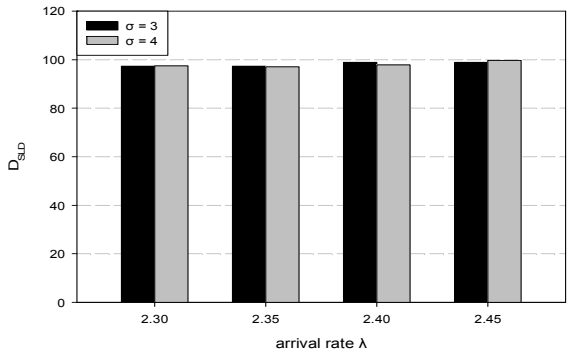

Fig. 6a. D_{SLD} vs *λ*, truncated normal distribution

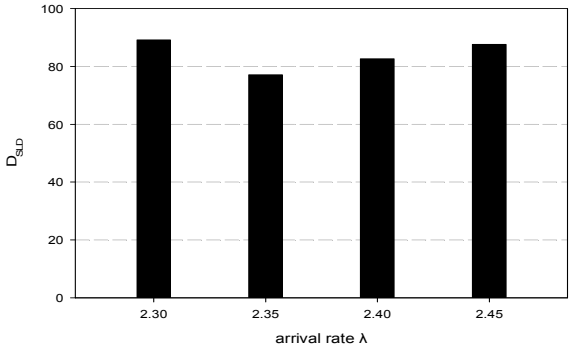

Fig. 6b. D_{SLD} vs *λ*, uniform distribution

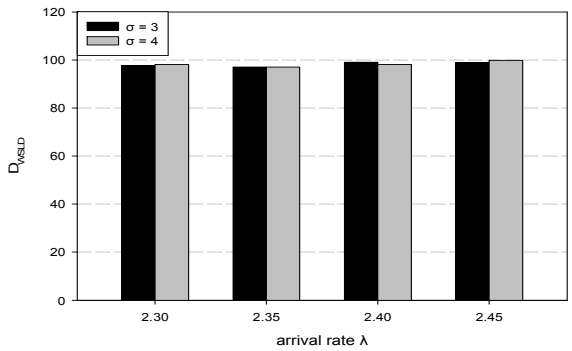

Fig. 7a. D_{WSLD} vs *λ*, truncated normal distribution

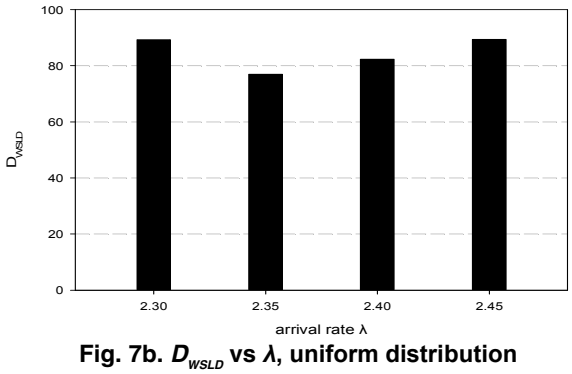

Fig. 7b. D_{WSLD} vs *λ*, uniform distribution

5. Related work

Related work which refers to gang scheduling also includes [5], [9] and [10]. Karatza [5] studied the performance of the AFCFS and the Largest Job First Served (LJFS) gang scheduling policies in the presence of critical sporadic jobs in a single cluster of distributed processors with no migrations.

In [9] we examine a migration strategy in the presence of high priority jobs. However, in [9] only the uniform distribution for gangs' degree of parallelism has been studied. This paper along with the uniform distribution also examines two cases of truncated normal distribution for the number of tasks per job.

Gang scheduling on top of existing batch scheduling policies with backfilling have been examined in [10]. This study [10] attempts to explore the possibilities of using synchronized time-sharing scheduling algorithms for a single small scale cluster.

The main contribution of this paper is the study of gang scheduling in cases of different distributions in the gangs' degree of parallelism in a multi-site system. To the best of the authors' knowledge, previous studies on the distribution of the gangs' degree of parallelism were not based on multi-site systems with migrations. Therefore, this study examines a system whose characteristics were not considered in any previous papers.

5. Conclusion and future work

This research examines the performance of gang scheduling implementing migrations. Simulation results are used to address performance issues associated with the variability on the gangs' degree of parallelism.

The simulation experiments indicate that the suggested migration scheme delivers superior results for various cases of gang size distributions. The simulation results also reveal that the gang size variability has a significant impact on the performance of gang scheduling when no migration scheme is applied.

In this paper, a homogeneous system consisting of two sites was used. For future work, we intend to study a larger scale heterogeneous distributed system which consists of a large number of sites.

References

[1] C. Du, X.H. Sun and M. Wu, "Dynamic Scheduling with Process Migration", In *Proc. of the Seventh IEEE International Symposium on Cluster Computing and the Grid*, IEEE Computer Society Washington DC, USA, 2007, pp. 92-99.

[2] 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, Berlin, 1997, Vol. 1291, pp. 238-261.

[3] S. Frechette, D.R. Avresky, "Method for Task Migration in Grid Environments", In *Fourth IEEE International Symposium on Network Computing and Applications*, IEEE Computer Society, Washington, DC, USA, 2005, pp. 49-58.

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

[5] H.D. Karatza, "The Impact of Critical Sporadic Jobs on Gang Scheduling Performance in Distributed Systems", *Simulation: Transactions of the Society for Modeling and Simulation International*, Sage Publications, Special Issue on Performance Evaluation of Computer and Telecommunication Systems, Vol. 84, No. 2-3, 2008, pp. 89-102.

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

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

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

[9] Z. Papazachos, and H. Karatza, "Performance Evaluation of Gang Scheduling in a Two-Cluster System with Migrations", to appear in the *8th International Workshop on Performance Modeling, Evaluation, and Optimization of Ubiquitous Computing and Network Systems,* Rome, Italy, May 25-29, 2009.

[10] H. Rajaei, M. Dadfar, P. Joshi, "Simulation of job scheduling for small scale clusters", In *Proc. of the 38th conference on Winter simulation*, Winter Simulation Conference, Monterey, CA, 2006, pp. 1195-1201.

[11] A. Streit, "Enhancements to the Decision process of the Self-Tuning dynP Scheduler", *In Job Sceduling Strategies for Parallel Processing*, Springer, Berlin, 2005, Vol. 3277, pp. 63-80.

[12] X.Y. Wang, Z.Y. Zhu, Z.H. Du and S.L. Li, "Multi-Cluster Load Balancing Based on Process Migration", In *Advanced Parallel Processing Technologies*, Springer, Berlin, 2007, Vol. 4847, pp. 100-110.

[13] Y. Zhang, H. Franke, J. Moreira, and A. Sivasubramaniam, "The Impact of Migration on Parallel Job Scheduling for Distributed Systems", In *Proc. of Euro-Par Parallel Processing*, Springer, Berlin, 2000, pp. 242-251.

[14] L Zheng, "A Task Migration Constrained Energy-Efficient Scheduling Algorithm for Multiprocessor Real-time Systems", In *International Conference on Wireless Communications, Networking and Mobile Computing (WiCom '07)*, 2007, pp. 3055-2058.