Gang Scheduling Performance on a Cluster of Non-Dedicated Workstations

Helen D. Karatza
Department of Informatics
Aristotle University of Thessaloniki
54006 Thessaloniki, Greece
karatza@csd.auth.gr

Abstract

Clusters of workstations have emerged as a cost-effective solution to high performance computing problem. To take advantage of any opportunities, however, effective scheduling techniques are necessary that enable parallel applications to dynamically share workstations with their owners. In this paper a special type of parallel applications called gangs are considered. Gangs are jobs that consist of a number of interacting tasks scheduled to run simultaneously on separate and distinct processors. A simulation model is used to address performance issues associated with gang scheduling on a non-dedicated workstation cluster for various workloads. Simulated results indicate that the relative performance of the gang scheduling policies that we examine depends on the workload characteristics.

1. Introduction

Clusters of workstations (COWs) are cost-effective platforms for parallel computation. The scheduling of parallel jobs on COWs is very important. However, most existing scheduling policies on multiprocessor / multi-computer systems are not appropriate because of the heterogeneous and non-dedicated features of many COWs.

Clusters are gaining acceptance not only in scientific applications that need supercomputing power, but also in domains such as databases, web service and multimedia, which place diverse Quality-of-Service (QoS) demands on the underlying system.

Resource management and scheduling on workstation clusters is complicated by the fact that the number of idle workstations available to execute parallel applications is constantly fluctuating.

Several resource management and scheduling issues arise when using COWS for parallel computing that have no counterpart in traditional parallel systems. Most of these issues arise due to the fact that workstations are typically "owned" by a user who may resent the presence of an external parallel computation on his or her computer. It has also been demonstrated that parallel applications need a dedicated environment to produce good performance. In order to keep workstations owners happy, it is necessary to ensure that parallel applications execute only on idle workstations.

It is not obvious how to effectively allocate nodes of a COW among competing jobs. One idea is to use gang scheduling. Gang scheduling allows tasks to interact efficiently by busy waiting, without the risk of waiting on a task that is currently not running. Without gang scheduling, tasks must block in order to synchronize, thus incurring context switch overhead.

Code to simultaneously schedule all tasks of each gang could be extremely complex, and require elaborate bookkeeping and global system knowledge. Because gang scheduling requires that no task execute unless all other gang member tasks also execute, some processors may be idle even when there are tasks waiting for processors.

With gang scheduling, there is always a one-to-one mapping between tasks and processors. Although the total number of tasks in the system may be larger that the number of processors, no gang contains more tasks than the number of available processors. We assume that all tasks within the same gang execute for the same amount of time, i.e. that the computational load is balanced between them.

A number of gang scheduling policies for distributed systems and multiprogrammed parallel systems have been proposed, each differing in the way resources are shared among the jobs [2], [3], [4], [5], [6], [8], [9], [10], and [11].

We study gang scheduling in an open queuing network that models a cluster of non-dedicated workstations incorporating workstation owner activities. The performance of two gang-scheduling policies are

Proceedings of the 35th Annual Simulation Symposium (SS'02)
1080-241X/02 $17.00 © 2002 IEEE

compared operating under various workloads. The allocation of a set of processors to a gang is assumed to be static and it does not change during its execution. To our knowledge, the analysis of this type of gang scheduling in non-dedicated workstations does not appear elsewhere in the research literature.

Non-dedicated clusters of workstations have been studied by [1]. However, that paper considers a different type of parallel job scheduling, since processor allocation of executing parallel jobs is reconfigured and changed.

The structure of this paper is as follows. Section 2.1 introduces the proposed system and workload models used to implement it, section 2.2 describes the scheduling policies and section 2.3 presents the metrics used to assess the performance of the scheduling policies. The model implementation and its input parameters are described in section 3.1, while simulation results are both presented and analysed in section 3.2. Finally, section 4 summarizes the paper and provides recommendations for further research.

2. Model and methodology

2.1 System and workload models

This paper uses a simulation model to address gang scheduling issues. An open queuing network model of a COW is considered. P = 16 homogeneous workstations are available, each serving its own queue. A high-speed network connects the distributed nodes. This is a representative model for many existing departmental COWs.

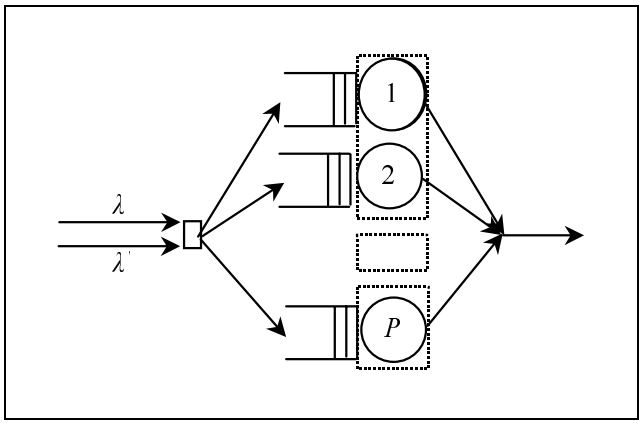

Figure 1. The queuing network model

The number of tasks in a job x is the job's *degree of parallelism* and it is represented as $t(x)$. If $p(x)$ represents the number of processors required by job x, then the following relationship holds:

$$1 \le t(x) = p(x) \le P$$

The number of tasks in a job x is called the "size" of job x. We call a job "small" ("large") if it requires a small (large) number of processors. The degree of parallelism is constant over the lifetime of system jobs.

We assume that the number of job tasks is uniformly distributed in the range of [1..P]. Therefore, the mean number of tasks per job is equal to the $\eta = (1+P)/2$.

Each task of job x is routed to a different processor for execution. The routing policy is based on the criteria "join the shortest queue" and is defined as follows:

Let:

* *n_assigned = number of tasks of job x that have been already assigned to a processor.*

Then:

* *n_assigned = 0;*
* *While n_assigned < t(x) do*
* *Begin*
* *n_processors = P – n_assigned;*
* *Select the shortest of the n_processor queues;*
* *n_assigned = n_assigned + 1;*
* *i = n_assigned;*
* *Assign the selected processor to the i^{th} task of job x;*
* *Remove this processor from the set of candidate processors for the next task assignment of job x*
* *End.*

Tasks in processor queues are examined in an order determined by the scheduling policy that is employed. Job x starts to execute only if all $p(x)$ processors assigned to it are available. Otherwise, all job x tasks wait in their assigned queues. When a job terminates execution, all processors assigned to it are released.

An important issue that arises in cluster environments is the need to handle owner activities on a subset of the nodes involved in a parallel computation. Therefore, a mechanism is needed to deal with the fact that underlying resources available to a parallel computation are changing.

The fluctuating processing capacity of a non-dedicated cluster of workstations poses several challenging problems for the job scheduler. The scheduler must give priority to owners, and at the same time provide good performance to multiple batch parallel applications that are trying to scavenge idle cycles from the cluster.

We consider that the job scheduler interrupts a parallel computation on a workstation upon detecting owner activity. The remaining processors that are assigned to the interrupted job can serve tasks of other gangs that are waiting at these processor queues.

Our research attempts to enhance system performance in terms of the mean response time for gangs assuming that owner jobs are served immediately.

When an owner job arrives during the execution of a gang task, all work that was performed on all tasks associated with that gang must be redone. The tasks of an interrupted gang are rescheduled for execution at the head of their assigned queues.

The technique used to evaluate the performance of the scheduling disciplines is experimentation using a synthetic workload simulation.

The workload considered here is characterized by three parameters:

- The distribution of gang service demand.
- The distribution of workstation owner job service demand.
- The mean inter-arrival time of gangs.
- The mean inter-arrival time of workstation owner jobs.

Gang and workstation owner job service demands are exponentially distributed with means of $1/\mu$ and $1/\mu'$ respectively.

We consider two arrival streams, one for gangs and one for workstation owners. The inter-arrival times of gangs and workstation owners are exponential random variables with means of $1/\lambda$, and $1/\lambda'$ respectively.

All notations used in this paper appear in Table 1.

In order to define an algorithm that assigns owner jobs to their respective processors, we consider the following mechanism:

On an owner arrival, one of the P workstations is selected randomly and if it is idle or it serves a task of a gang, we assume the job that just arrived belongs to this workstation owner. Otherwise, another workstation is selected randomly from among the remaining P-1 processors and so on. In the experiments that we conduct, values of λ' are chosen that do not result in further owner arrivals in cases where all workstations are occupied by their owners.

2.2 Job scheduling policies

It is assumed that the scheduler has comprehensive information available when making decisions, i.e. it knows the exact number of processors required by each job. The following two scheduling strategies are employed in our simulations:

- ***Adapted-First-Come-First-Served* (*AFCFS*)**. This method attempts to schedule a job whenever processors assigned to its tasks are available. When there are not enough processors available for a large job whose tasks are waiting in the front of the queues, AFCFS policy schedules smaller jobs whose tasks are behind the tasks of the large job.

  One major problem with this scheduling policy is that it tends to favor jobs requiring a smaller number of processors and thus may increase fragmentation of the system.

- ***Largest-Gang-First-Served* (*LGFS*)**. With this policy tasks are placed in increasing job size order in processor queues (tasks that belong to larger gangs are placed at the head of queues). All tasks in queues are searched in order, and the first jobs whose assigned processors are available begin execution.

  This method tends to improve the performance of large, highly parallel jobs at the expense of smaller jobs, but in many computing environments this discrimination is acceptable, if not desirable. For example, supercomputers often run large, highly parallel jobs that cannot run elsewhere.

When a gang is interrupted due to owner arrival, all tasks of that gang are resubmitted for execution as the leading tasks in their assigned queues. They wait at the head of the ready queues until all processors allocated to this job are available. The remaining processors assigned to the interrupted job execute tasks of other jobs waiting in their queues.

When the owner task terminates, the interrupted job likely does not resume execution immediately as some of processors assigned to it may be working on other jobs. Those jobs will not terminate at the same time so it is impossible for the interrupted job to use them efficiently.

Note also that when an owner arrives at a workstation, it is not only the tasks of this workstation queue that are delayed, but also tasks in other workstation queues that have a sibling task waiting for service in the same queue. The larger the owner job service, the higher the probability that some gangs will have long delay if one of their tasks is at that workstation queue.

2.3 Performance metrics

We define gang r*esponse time* as the interval from the dispatching of this job tasks to processor queues to service completion of this job (time spent in processor queues plus time spent in service).

Parameters used in simulation computations (presented later) are shown in Table 1.

Table 1. Notations

P	number of processors
U	mean processor utilisation
μ	mean gang service rate
$1/\mu$	mean gang service demand
μ'	mean workstation owner job service rate
$1/\mu'$	mean workstation owner job service demand
λ	mean arrival rate of gangs
λ'	mean arrival rate of workstation owner jobs
RT	mean response time of gangs
RT ratio	the ratio of RT in the LGFS case over RT in the AFCFS case

The relative performance of the two scheduling policies that we consider is measured by the *RT ratio*.

3. Simulation results and discussion

3.1 Model implementation and input parameters

The queuing network model described above is implemented with discrete event simulation [7] using the independent replication method.

For every mean value, a 95% confidence interval is evaluated. All confidence intervals are less than 5% of the mean values.

With regard to gangs and workstation owners arrival rates, two set of experiments were carried out:

- In the first set, for gang mean inter-arrival time $1/\lambda = 1$, and for $1/\mu = 1/\mu' = 1$, we vary the mean inter-arrival time of the workstation owners as follows: $1/\lambda' = 1$, 0.8, 0.6, and 0.4. We are therefore able to consider cases where owner activities occur with different frequencies over time.

- In the second set of experiments, we consider $1/\mu = 1$, and we study three cases for the mean inter-arrival time of gangs: $1/\lambda = 1$, 0.8, and 0.6. In each of these cases we study the following combinations of mean inter-arrival time – mean service demand of workstation owners: ($1/\lambda' = 1$, $1/\mu' = 1$), ($1/\lambda' = 2$, $1/\mu' = 2$), ($1/\lambda' = 4$, $1/\mu' = 4$), and ($1/\lambda' = 8$, $1/\mu' = 8$). We are therefore able to study for various cases of λ, the impact of the frequency and the duration of owner activities on gang's performance.

It should be noted that there are on average 8.5 tasks per parallel job ($(P+1)/2$). So, if we do not consider owner jobs and all processors are busy due to gang service, then an average of P / 8.5 = 1.88235 parallel jobs can be served each unit of time. This implies that we should choose a $\lambda < 1.88235$.

However, due to owner activity, the number of processors that are available to gang service is $[P - (\lambda'/\mu')]$. Therefore, we have to choose a value of λ for which the following relationship holds:

$$\lambda < [P - (\lambda'/\mu')] / 8.5$$

3.2 Performance analysis

Due to space considerations, the following partial results are presented. The results describe overall relative performance of the two different scheduling policies very accurately.

Figures 2-5 are RT and *RT ratio* versus $1/\lambda'$. Figures 6-7 are U versus $1/\lambda'$. Tables 2-5 present U in all cases that we examined. Simulation results indicate the following:

a. Performance with regard to mean response time of gangs

First set of experiments: Figures 2 - 3 show that for all arrival rates of workstation owner jobs the LGFS method yields lower mean response time than AFCFS. The difference in performance between the two policies is higher in the case where the inter-arrival time of the workstation owner jobs is equal to the inter-arrival time of gangs. As the arrival rate of workstation owner jobs increases, the difference in performance decreases and tends to be constant for $1/\lambda' < 0.8$. This is due to the following: The higher the arrival rate of workstation owner jobs is, the greater is the possibility that fewer processors are available for gang service. Therefore, the potential of the LGFS policy is not completely exploited, as large gangs cannot find enough idle processors to serve them.

We can also observe a sharp increase in the response time of gangs where $1/\lambda' < 0.8$. In this case, both scheduling policies yield very high mean gang response time as compared with the mean gang service time. This means that the mean number of processors that are available to serve gangs has to be greater than 14.33, in order for gangs to have acceptable performance.

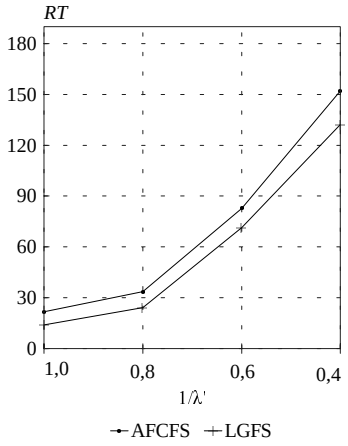

Figure 2. *RT* versus 1/*λ′* (1/*λ* = 1, *μ* = *μ′* = 1)

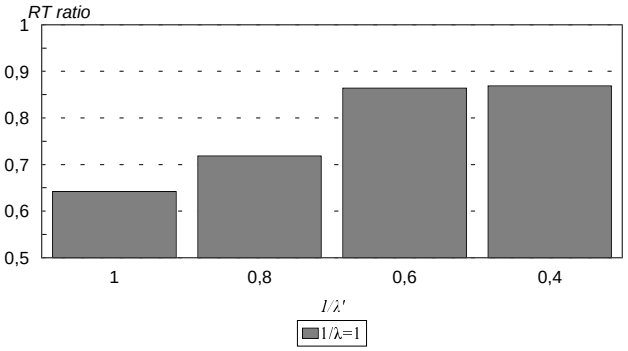

Figure 3. *RT ratio* versus 1/*λ′*, (1/*λ* = 1, *μ* = *μ′* = 1)

Second set of experiments: In these experiments we consider various cases where the mean inter-arrival rate of workstation owner jobs is equal to their mean service rate. These cases are studied for different gang mean inter-arrival time. Figures 4 - 5 show that in most cases of gang mean inter-arrival time that we examined, overall performance is superior with the LGFS method in terms of mean response time.

However, when 1/λ = 0.6 (λ =1.67), with either scheduling method *RT* is very high as compared with the 1/λ =1, and 1/λ = 0.8 cases. This is because more gangs delay in processor queues when mean gang arrival rate is equal to 1.67 than when it is smaller. Further to this, since workstation owner jobs have higher priority as compared to gangs, when they block a gang they not only influence the performance of the current gang, but also affect the performance of subsequent gangs. When a workstation occupied by its owner is released, some of the processors assigned to the interrupted job may already have served other jobs. Those jobs will not finish at the same time, so their processors are not used efficiently.

In Figure 5 we observe that for all λ', the superiority of the LGFS method over the AFCFS policy increases as 1/λ decreases from 1 to 0.8. This is because the advantages of the LGFS case are exploited better when there are a sufficient number of gangs in the queues so that they can be selected according to the LGFS criteria. However, the difference in performance decreases with a further decrease of 1/λ from 0.8 to 0.6. This is because high loads cause queuing delays of gangs with either scheduling method. Especially for 1/λ = 0.6, and 1/λ' = 1/μ'= 2 the two policies produce almost the same *RT*. Therefore, if overhead is considered, in this case AFCFS is preferred as it results in less overheard.

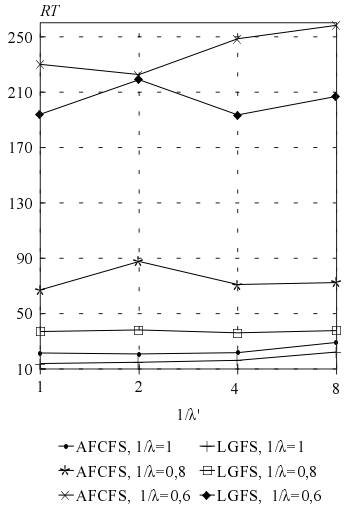

Figure 4. *RT* versus 1/*λ′*, (*μ* = 1, *μ′* = *λ′*)

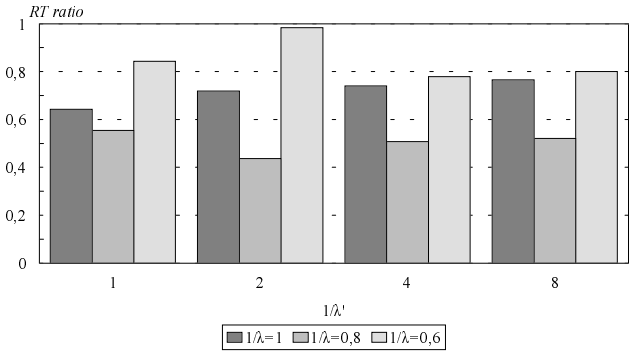

Figure 5. *RT ratio* versus 1/*λ′*, (*μ* = 1, *μ′* = *λ′*)

b. Performance with regard to mean processor utilization

In most cases the mean processor utilization is higher with the LGFS policy than with AFCFS (Tables 2-5, Figures 6-7). However, with both policies, part of the

processor utilization is comprised of repeated gang work, caused by workstation owner arrivals. The repeat work depends on the number of tasks in the gang, the task service demand, and the work that has already been accomplished by the gang at the moment of interruption. It is possible for a gang to be interrupted many times over the period of its execution. This would be caused by multiple owner arrivals at different workstations serving tasks of this gang.

In Tables 4-5, and in Figure 7 it is shown that for any λ' the largest difference in processor utilization between the two policies appears for $1/\lambda = 0.6$. However, as we have already noted, an increase in processor utilization does not necessarily mean performance improvement.

Table 2. Mean processor utilization, AFCFS case, ($1/\lambda = 1$, $\mu = \mu' = 1$)

	$1/\lambda'= 1$	$1/\lambda'= 0.8$	$1/\lambda'= 0.6$	$1/\lambda'= 0.4$
$1/\mu = 1$		$1/\mu'= 1$		
$1/\lambda = 1$	0.588	0.588	0.602	0.598

Table 3. Mean processor utilization, LGFS case, ($1/\lambda = 1$, $\mu = \mu' = 1$)

	$1/\lambda'= 1$	$1/\lambda'= 0.8$	$1/\lambda'= 0.6$	$1/\lambda'= 0.4$
$1/\mu = 1$		$1/\mu'= 1$		
$1/\lambda = 1$	0.577	0.610	0.617	0.617

Table 4. Mean processor utilization, AFCFS case, ($\mu = 1$, $\mu' = \lambda'$)

	$1/\lambda'= 1$	$1/\lambda'= 2$	$1/\lambda'= 4$	$1/\lambda'= 8$
$1/\mu = 1$		$\mu' = \lambda'$		
$1/\lambda=1.0$	0.588	0.581	0.579	0.587
$1/\lambda=0.8$	0.681	0.701	0.693	0.692
$1/\lambda=0.6$	0.739	0.746	0.746	0.750

Table 5. Mean processor utilization, LGFS case, ($\mu = 1$, $\mu' = \lambda'$)

	$1/\lambda'= 1$	$1/\lambda'= 2$	$1/\lambda'= 4$	$1/\lambda'= 8$
$\mu = 1$		$\mu' = \lambda'$		
$1/\lambda=1.0$	0.577	0.586	0.573	0.587
$1/\lambda=0.8$	0.700	0.706	0.701	0.699
$1/\lambda=0.6$	0.826	0.837	0.835	0.833

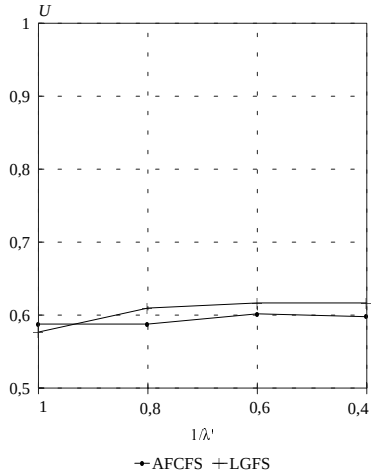

Figure 6. *U* versus $1/\lambda'$, ($1/\lambda = 1$, $\mu = \mu' = 1$)

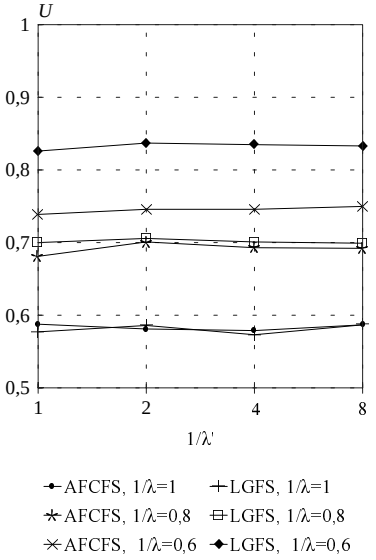

Figure 7. *U* versus $1/\lambda'$, ($\mu = 1$, $\mu' = \lambda'$)

4. Conclusions and further research

This paper examines the performance of two gang scheduling policies in a non-dedicated workstation cluster. A simulation model is used to address performance issues associated with gang scheduling for various workloads.

The following is a summary of the simulation results:

- In most cases that we examined, the Largest-Gang-First-Served (LGFS) method outperforms the Adapted – First – Come – First – Served (AFCFS) policy.
- The relative performance of the two gang scheduling policies depends on the workload characteristics.
- With respect to scheduling overhead, in those cases where performance of the two methods does not differ significantly, the AFCFS method is preferred, as it is easier to implement.

This paper is a case study. It can be extended to the case where after an owner departs, the interrupted parallel job can restart from an intermediate checkpoint. This requires the periodic saving of each task's internal state.

References

[1] A. Chowdhury, L.D. Nicklas, S.K. Setia, and E.L. White, "Supporting Dynamic Space-sharing on Clusters of Non-dedicated Workstations", In *Proceedings of the 17th International Conference on Distributed Computing Systems (ICDS '97)*, IEEE, Baltimore, Maryland, 28-30 May 1997, pp. 149-158.

[2] D.G. Feitelson, and L. Rudolph, "Parallel Job Scheduling: Issues and Approaches", In *Job Scheduling Strategies for Parallel Processing*, *Lecture Notes in Computer Science*, Springer-Verlang, Berlin, Germany, 1995, Vol. 949, pp. 1-18.

[3] D.G. Feitelson, and L. Rudolph, "Evaluation of Design Choices for Gang Scheduling Using Distributed Hierarchical Control", *Journal of Parallel and Distributed Computing*, Academic Press, New York, USA, 1996, Vol. 35, pp. 18-34.

[4] D.G. Feitelson, and M.A. Jette, "Improved Utilisation 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. 238-26.

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

[6] H.D. Karatza, "Gang Scheduling and I/O Scheduling in a Multiprocessor System", In *Proceedings of SPECTS'2K, 2000 SCS Symposium on Performance Evaluation of Computer and Telecommunication Systems*, SCS, Vancouver, BC, Canada, July 16-20, 2000, pp. 245-252.

[7] Law, A., and D. Kelton, *Simulation Modelling and Analysis*, McGraw-Hill, New York, 1991.

[8] P.G. Sobalvarro, and W.E. Weihl, "Demand-based Coscheduling of Parallel Jobs on Multiprogrammed Multiprocessors", In *Job Scheduling Strategies for Parallel Processing*, *Lecture Notes in Computer Science*, Springer-Verlang, Berlin, Germany, 1995, Vol. 949, pp. 106-126.

[9] M.S.Squillante, F. Wang, and M. Papaefthymioy, "Stochastic Analysis of Gang Scheduling in Parallel and Distributed Systems", *Performance Evaluation*, Elsevier, Amsterdam, Holland, 1996, Vol. 27&28 (4), pp. 273-296.

[10] F. Wang, M. Papaefthymiou, and M.S. Squillante, "Performance Evaluation of Gang Scheduling for Parallel and Distributed Systems", In *Job Scheduling for Parallel Processing*, *Lecture Notes in Computer Science*, Springer-Verlang, Berlin, Germany, 1997, Vol. 1291, pp. 184-195.

[11] 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 2000, pp. 242-251.