Scheduling a Job Mix in a Partitionable Parallel System

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

Ralph C. Hilzer
Computer Science Department
California State University, Chico
Chico, California 95929-0410 USA
hilzer@ecst.csuchico.edu

Abstract

Efficient scheduling of jobs on parallel processors is essential for good performance. However, design of such schedulers is challenging because of the complex interaction between system and workload parameters. This paper studies the performance of a partitionable parallel system in which job scheduling depends on job characteristics. Jobs consist of different number of tasks and are characterized as sequential or parallel depending on whether the tasks are processed sequentially on the same processor or at different processors. Jobs that consist of parallel tasks are called gangs, that is, they have to be scheduled to execute concurrently on processor partitions, where each task starts at the same time and computes at the same pace. The goal is to achieve good performance of sequential and parallel jobs. The performance of different scheduling schemes is compared over various workloads. Simulated results indicate that sequential jobs should not arbitrary overtake the execution of parallel jobs.

1. Introduction

The scheduling of jobs on processors of a partitionable parallel machine is an important and challenging area. The allocation and management of resources for these systems is fundamental to sustaining and improving the benefits of multiprocessing [3], and [9].

Jobs usually have different characteristics. For example, some jobs are comprised of sequential tasks, while other jobs consist of multiple tasks that can be run in parallel on different processors. It is not possible to efficiently execute all jobs with one type of scheduler. It is crucial to apply the proper scheduling strategy to jobs according to their characteristics.

In this paper we compare the effect of different job scheduling policies operating under various workloads. The goal is to achieve high system performance while maintaining fairness in terms of sequential and parallel job execution.

Our work involves a shared memory system with 128 processors. We use a scalable, coherent shared address space (SAS) multiprocessing system that has been the focus of many other studies. Over the last decade, a number of hardware cache-coherent, non-uniform memory access architectures (so-called hardware-DSM or CC-NUMA machines) have been implemented and shown to perform well at a moderate scale of about 32 processors. In fact, such machines are quickly becoming the dominant form of tightly coupled multiprocessor machines built by commercial vendors.

An open question is how scalable these architecture configurations are to larger processor numbers. In [7], the performance of a wide range of SAS parallel applications on a 128-processor hardware cache-coherent machine (the SGI Origin2000) is studied. It is demonstrated that scalable performance is indeed achieved with this programming model over a wide range of applications, including the challenging of kernels like FFT.

This study considers a partitionable parallel processing system where the partitions are subsystems allocated to independent jobs. Some of the jobs are sequential while the remainder are parallel. Sequential tasks run on the same processor. Parallel tasks execute concurrently on a set of processors. The parallel tasks start at essentially the same time, co-ordinate their execution, and compute at the same pace. This type of resource management is called “coscheduling” or “gang scheduling” and is extensively studied in the literature of parallel and distributed systems [1], [2], [4], [5], [6], [8], [10], [11], [13], [14], and [15].

Jobs begin execution only if enough idle processors are available to handle them. However, a scheduling policy is needed to determine which job is to be mapped to the available processors. In order to achieve fairness in terms of individual job class service, sequential jobs should not be arbitrarily scheduled ahead of the parallel jobs.

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

The design choices considered in this paper include different ways to schedule sequential jobs and gangs for service on the system processors. The performance of different scheduling policies is compared for various workloads. The authors have found no evidence that this type of scheduling has previously been applied to this system operating under these workload models.

The scheduling of sequential jobs and gangs in a partitionable parallel system was studied earlier in [10]. However, in that paper the system is a closed queuing network with a fixed number of jobs. Furthermore, while that paper examines only one case relating to the number of tasks per parallel and sequential job, this one examines various cases.

The technique used to evaluate the performance of the scheduling disciplines is experimentation using a synthetic workload simulation. In studies like this, it is usually necessary to use synthetic workloads because real workloads cannot be simulated efficiently enough and real systems with actual workloads are not available for experimentation. Also, useful analytic models are difficult to derive because subtleties that exist between various disciplines are difficult to model and because the workload model is quite complex.

This paper is an experimental study in that the results are obtained from simulation studies instead of from the measurements of real systems. Nevertheless, the results presented are of practical value. All of the algorithms are practical in that they can be implemented. Although we do not derive absolute performance values for specific systems and workloads, we do study the relative performance of the different algorithms across a broad range of workloads and analyze how changes in the workload can affect performance.

The structure of this paper is as follows. Section 2.1 specifies system and workload models, section 2.2 describes scheduling strategies, and section 2.3 presents the metrics employed while assessing performance of the scheduling policies. Experimental methodology is described in section 3, while experimental results are presented and analyzed in section 4. Section 5 contains conclusions and suggestions for further research.

2. Model and methodology

2.1 System and workload models

An open queuing network model is considered that consists of $P = 128$ parallel homogeneous processors.

All processors share a single queue (memory). The effects of the memory requirements and the communication latencies are not represented explicitly in the system model. Instead, they appear implicitly at job execution time.

The configuration of the model is shown in Figure 1.

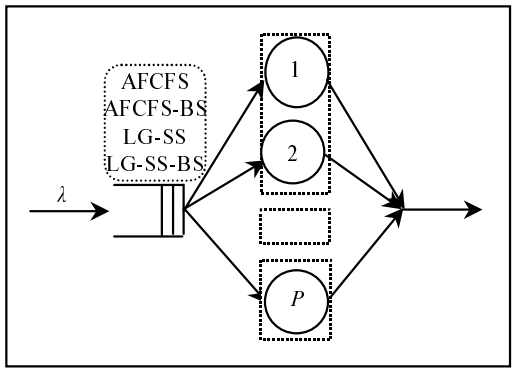

Figure 1. The queuing network model

The queuing network model is implemented with discrete event simulation. We evaluate the performance of job scheduling algorithms under various workload models, each of which has certain characteristics relating to the:

- Number of job tasks.
- Interdependence of job tasks.
- The distribution of job inter-arrival time.
- The distribution of task service demand.

- **Number of tasks per job**

We consider that every job x consists of t_x tasks where $1 \le t_x \le P$. Therefore, we bind the number of tasks per job to the number of processors in the system. The number of tasks that make up job x is called the "size" of job x. A job is said to be "small" (or "large") if it consists of a small (or large) number of tasks. The number of processors required by job x is represented as $p(x)$. It is obvious that $t_x \ge p(x)$.

The analysis of real workload logs, collected from many large-scale parallel computers used in production, shows that the percentage of small jobs, with a small number of tasks, is higher than large jobs, with a large number of tasks. For this reason, we examine the following distribution for the number of tasks per job.

Uniform-log model

Job size is an integer calculated by 2^i within the range $[1, P]$, where i is an integer in the range $[0, \log P]$. The probability of each value is uniform. Therefore, in our model job sizes are 1, 2, 4, 8, 16, 32, 64, 128.

- **Interdependence among tasks of a job**

We consider that the tasks of a job belong to one of the following two categories:

Sequential tasks

Job tasks have precedence constraints and have to be processed in sequence on the same processor. Therefore, it holds that $p(x) = 1$ and $t_x \geq p(x)$.

Gangs

Those jobs consisting of tasks that execute concurrently on processor partitions, where each task starts at the same time and computes at the same pace, are called *gangs*. Gang scheduling executes a set of tasks simultaneously on a set of processors. It allows tasks to interact efficiently by busy waiting, without the risk of waiting for a task that currently is not running. Without gang scheduling, tasks must block in order to synchronize, thus incurring context switch overhead. At any time, there is a one-to-one mapping between tasks and processors. We assume that all tasks within the same gang execute for the same amount of time, i.e., the computational load is balanced among them. Each job begins execution only when a sufficient number of idle processors are available to meet its needs. It holds that: $t_x = p(x)$.

Gangs x_1, x_2, ... x_m can be executed simultaneously with s sequential jobs, where $0 \leq s < P$, if and only if the following relation holds:

$$s + \sum_{i=1}^{m} p(x_i) \leq P.$$

In our model, jobs that consist of $1 \leq n \leq N_{max}$ tasks are sequential, while jobs that consist of $N_{max} < n \leq P$ tasks are gangs. We examine different N_{max} values.

- ***Distribution of job inter-arrival time***

We consider that job inter-arrival times are exponential random variables with a mean of $1/\lambda$.

- ***Distribution of task service demand***

Service demands of tasks of sequential and parallel jobs are exponentially distributed with a mean of $1/\mu$.

The sequence that jobs in the queue are served depends on the scheduling policy. Fairness is required across competing jobs.

Next we describe the scheduling strategies selected for this study. As is the case with most studies, we assume that scheduling overhead is negligible. We also assume that the scheduler has perfect information when making decisions, i.e. it knows the exact number of processors required by all jobs in the queue.

2.2 Job scheduling policies

Adapted First Come First Served (AFCFS)

When a processor or a set of processors become idle, all jobs in the queue are examined in sequence for execution on the available processors. One major problem with AFCFS is that it may introduce large queuing delays within gangs since it favors sequential job tasks. This problem is compensated for in the following method.

AFCFS-Blocking of Sequential Jobs (AFCFS-BS)

When a job leaves, if the first job in the queue is a gang and it can be scheduled, then all other jobs in the queue are examined for execution on the remaining available processors. If the gang cannot start on the available processors, then only other gangs in the queue are examined. Sequential jobs are blocked. When a sequential job arrives and the first job in the queue is a gang, the sequential job is blocked.

Largest Gang First Served / Shortest Sequential Job First Served (LG-SS)

With this method, gangs are placed in processor queues in the order of increasing job size (larger gangs are moved to the head of the queue). On the other hand, it is well known that when coscheduling is not required for tasks of jobs, shortest service time first is the optimal method. However, in most cases, advance knowledge of task service time is not available. For this reason, we consider the number of tasks of a sequential job as an indication of the cumulative service time of its constituent tasks. Sequential jobs (groups of sequential tasks) are placed in the queue in decreasing number of task order.

LG-SS-Blocking of Sequential Jobs (LG-SS-BS)

This is a version of the LG-SS policy where the blocking of sequential jobs occurs in a manner similar to the AFCFS –BS case.

2.3 Performance metrics

Response time of a job is the time interval from the arrival of that job at the processors queue to the service completion time for that job (i.e., time spent in the processors queue plus job service time).

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

Table 1: Notations

μ	mean task service rate
$1/\mu$	mean task service demand
λ	mean job arrival rate
$1/\lambda$	mean job inter-arrival time
RT_s	mean response time of sequential jobs
RT_g	mean response time of parallel jobs
MRT_s	maximum response time of sequential jobs
MRT_g	maximum response time of parallel jobs
W	mean waiting time of jobs
N_{max}	maximum number of tasks per sequential job

RT_s and RT_g represent the performance of sequential and parallel jobs respectively. MRT_s and MRT_g represent fairness in terms of individual job class service. W represents the overall job performance.

3. Experimental methodology

The queuing network model is simulated with discrete event simulation models [12] using the independent replication method. For every mean value, a 95% confidence interval is computed. All confidence intervals are within 5% of the mean values.

In the simulation experiments we defined mean task service demand and mean inter-arrival time as follows:

$$1/\mu = 1, \text{ and } 1/\lambda = 0.280$$

The value 0.280 was chosen because the processors average 31.875 tasks per job, as the following relation holds:

$$(1/(\log P + 1)) * \sum_{i=0}^{\log P} 2^i = 31.875$$

When all processors are busy, an average of 4.0157 jobs are served each unit of time. This implies that the arrival rate has to be less than 4.0157, which means $1/\lambda > 0.249$, so that the processors queue will not be saturated. For this reason we choose a larger mean inter-arrival time $1/\lambda = 0.280$ that results in traffic intensity equal to 0.889.

We vary the maximum number of tasks per sequential job as follows:

$$N_{max} = 2^3, 2^2, 2^1$$

which means that 25%, 37.5% and 50% of the jobs respectively are sequential. We are therefore able to study the performance of the scheduling policies for different job mix cases.

4. Experimental results and discussion

The following results are presented:

- Figure 2 is W versus N_{max} for all cases that we examined.
- Figure 3 is the RT_s and RT_g versus N_{max} for the AFCFS and AFCFS-BS cases.
- Figure 4 is the MRT_s and MRT_g versus N_{max} for the AFCFS and AFCFS-BS cases.
- Figure 5 is the RT_s and RT_g versus N_{max} for the LG-SS and LG-SS-BS cases.
- Figure 6 is the MRT_s and MRT_g versus N_{max} for the LG-SS and LG-SS-BS cases.
- Figure 7 is the ratio RT_g / RT_s in the AFCFS, AFCFS-BS, LG-SS, and LG-SS-BS cases.
- Figure 8 is the ratio MRT_g / MRT_s in the AFCFS, AFCFS-BS, LG-SS, and LG-SS-BS cases.

The results demonstrate the following:

The mean processor utilization varies in the N_{max} = 8, 4, and 2 cases over the ranges of 0.53-0.81, 0.74-0.86, and 0.80-0.88 respectively. Except for where AFCFS N_{max} = 2, utilization is higher in all other cases when blocking sequential tasks than in the non-blocking cases.

Therefore, the blocking of sequential jobs improves system performance. The reason is apparent. There are more opportunities for sequential jobs to start than for gangs when a processor becomes available. Also, when a sequential job begins execution, it occupies a processor on average for a longer time interval than a gang task. Therefore, when a gang is waiting for available processors, it cannot access this processor for this time period. This may keep some processors idle even though there are jobs waiting in the queue.

Regarding the mean waiting time of all jobs, the AFCFS-BS and LG-SS-BS methods perform better than the AFCFS and LG-SS methods respectively. This is because the non-blocking case accumulates gangs in the queue and suffers long delays. These gang delays affect the mean waiting time more seriously than the delays of sequential jobs do in the blocking cases. In all cases the

mean waiting time is lower in the AFCFS-BS case than in the LG-SS-BS case.

Regarding the mean response time of sequential jobs and of gangs, in all cases the blocking of sequential jobs increases RT_s and decreases RT_g (Figures 3, and 5). In the non blocking cases, RT_s is smaller than RT_g (with one exception in the LG-SS case where RT_s and RT_g are almost the same). The opposite happens in the blocking cases where RT_s is larger than RT_g. The RT_g / RT_s ratio in each one of the AFCFS, AFCFS-BS, LG-SS, and LG-SS-BS cases is depicted in Figure 7. It is apparent from this Figure that in the non blocking cases, RT_g is larger than RT_s in a larger degree in the AFCFS case than in the LG-SS case. In the blocking cases where as we already noted that $RT_g < RT_s$, RT_g is smaller than RT_s in a larger degree in the LG-SS-BS case than in the AFCFS-BS case.

Figure 8 represents the ratios MRT_g / MRT_s in the AFCFS, AFCFS-BS, LG-SS, and LG-SS-BS cases. In the AFCFS case MRT_g is larger than MRT_s. In the AFCFS-BS case MRT_g is also larger than MRT_s, but in a lesser degree than in the AFCFS case. In the LG-SS case for N_{max} = 4, MRT_g is larger than MRT_s, while for N_{max} = 8, and N_{max} = 2, MRT_g is slightly smaller than MRT_s. In the LG-SS-BS case MRT_g is smaller than MRT_s.

From Figures 4 and 6, and also from Figure 8 we observe that the AFCFS-BS policy is generally fairer to sequential and parallel job classes than the LG-SS-BS policy. This is because the AFCFS-BS method in most cases yields lower maximum response times of both sequential jobs and gangs than the LG-SS-BS policy does (the only exception is at N_{max} = 8 where gangs have lower MRT_g in the LG-SS-BS case than in the AFCFS-BS case). Further to this, the maximum response times of sequential jobs and gangs differ between each other in a lesser degree in the AFCFS-BS case than in the LG-SS-BS case.

For N_{max} = 8 the mean response time of parallel jobs is lower in the LG-SS-BS case than in the AFCFS-BS case (Figures 3, and 5). For N_{max} = 4 the mean response time of parallel jobs is almost the same in the LG-SS-BS and AFCFS-BS cases. However, for N_{max} = 8, and N_{max} = 4 the sequential job mean response time is much larger in the LG-SS-BS case than it is in the AFCFS-BS case. Furthermore, as we have already noted, the LG-SS-BS case sequential jobs present very high MRT_s as compared to the MRT_s of the AFCFS-BS case. On the other hand, the difference in the maximum response time of gangs in the AFCFS-BS and LG-SS-BS cases is small as compared to the difference in the maximum response time of the sequential jobs in the corresponding cases.

For N_{max} = 2, the mean response time and maximum response time of parallel and sequential jobs is lower in the AFCFS-BS case than in the LG-SS-BS case. However, it should be noted that sequential jobs comprise the 25% only of the total number of jobs in this case and also they have only one or two sequential tasks. Therefore, it is apparent that they affect the performance of gangs to a lesser degree than in cases where they represent a larger percentage of the job mix.

The results do not take overhead into account relating to the complexity of the scheduling policy employed. The AFCFS-BS method is easier to implement and therefore involves less overhead than the LG-SS-BS method.

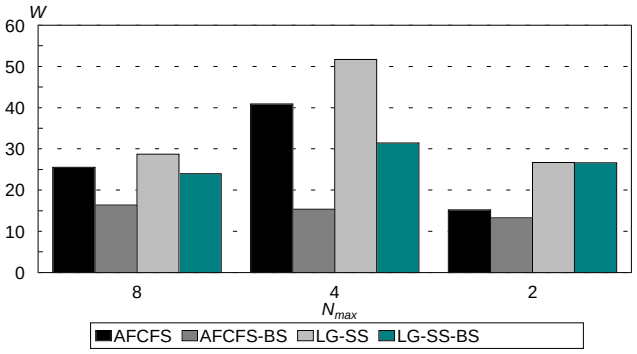

Figure 2. *W* versus N_{max}

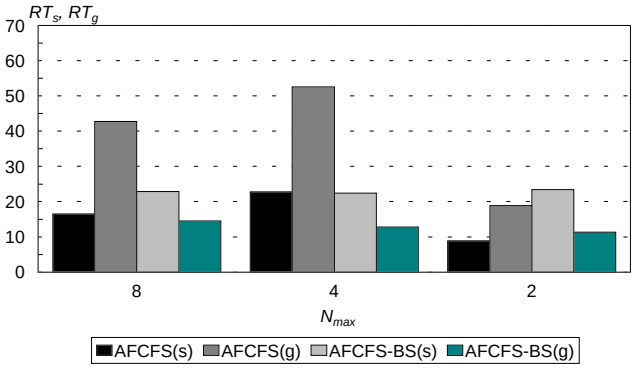

Figure 3. RT_s and RT_g versus N_{max} for the AFCFS and AFCFS-BS cases

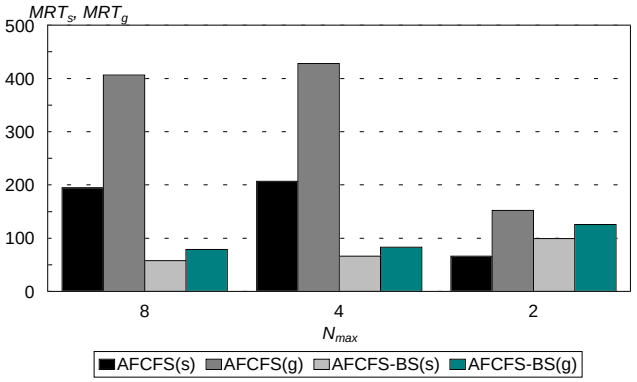

Figure 4. MRT_s and MRT_g versus N_{max} for the AFCFS and AFCFS-BS cases

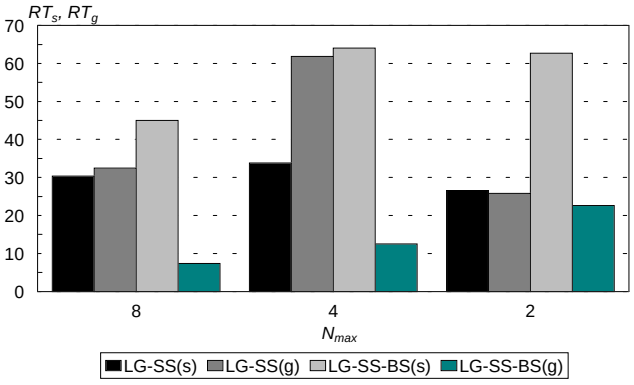

Figure 5. RT_s and RT_g versus N_{max} for the LG-SS and LG-SS-BS cases

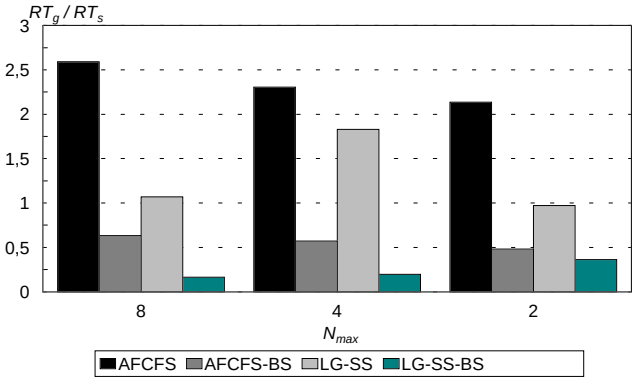

Figure 7. Ratio RT_g / RT_s versus N_{max}

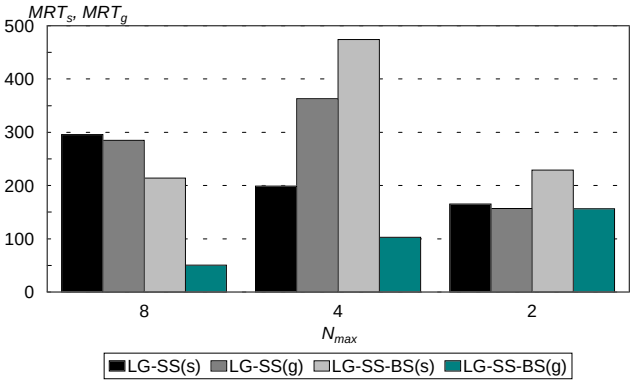

Figure 6. MRT_s and MRT_g versus N_{max} for the LG-SS and LG-SS-BS cases

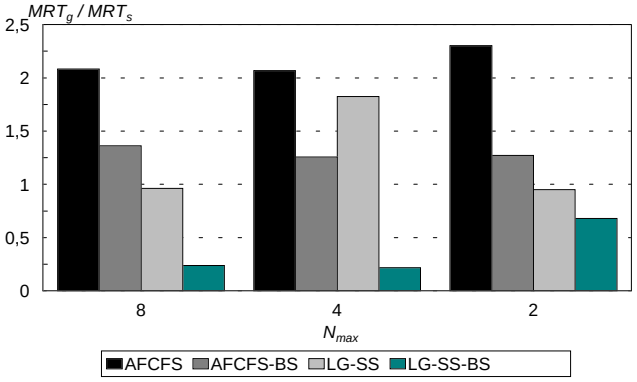

Figure 8. Ratio MRT_g / MRT_s versus N_{max}

5. Conclusions and further research

This paper studies the scheduling of sequential and parallel jobs in a partitionable parallel processing system. We use simulation as the means to generate the results used to compare different configurations.

Four scheduling policies are considered (AFCFS, LG-SS, AFCFS-BS and LG-SS-BS). Their performance is simulated and the results are compared for different numbers of sequential and parallel jobs. The goal is to assure fairness in individual job class service. Simulation results indicate the following:

- The blocking of sequential jobs improves overall performance and also protects gangs from excessive delays.

- With respect to the types of job mix that we examined, the AFCFS-BS method should be used instead of LG-SS-BS. This is because AFCFS-BS is easier to implement and in most cases performs better than LG-SS-BS regarding overall job performance and fairness in individual job class service.

This is a case study. Further experimentation is necessary to examine other cases which involve parallel jobs which consist of independent tasks that can execute at any processor and in any order along with sequential jobs and gangs.

References

[1] K. Aida 2000, "Effect of Job Size Characteristics on Job Scheduling Performance", In *Job Scheduling Strategies for Parallel Processing, Lecture Notes in Computer Science*, D.G. Feitelson and L. Rudolph (eds.), Springer-Verlang, Berlin, Germany, 1997, Vol. 1911, pp. 1-10.

[2] K. Aida, H. Kasahara, and S. Narita, "Job Scheduling Scheme for Pure Space Sharing among Rigid Jobs", In *Job Scheduling Strategies for Parallel Processing, Lecture Notes in Computer Science*, D.G. Feitelson and L. Rudolph (eds.), Springer-Verlang, Berlin, Germany, 1998, Vol. 1459, pp. 98-121.

[3] L.W. Dowdy, E. Rosti, G. Serazzi, and E. Smirni, "Scheduling Issues in High-Performance Computing", *Performance Evaluation Review*, ACM, New York, USA, Vol. 26 (4), 1999, pp. 60-69.

[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*, D.G. Feitelson and L. Rudolph (eds.), Springer-Verlang, Berlin, Germany, 1997, Vol. 1291, pp. 238-261.

[5] D.G. Feitelson, and L. Rudolph, "Parallel job scheduling: issues and approaches". In *Job Scheduling Strategies for Parallel Processing, Lecture Notes in Computer Science*, D.G. Feitelson and L. Rudolph (eds.), Springer-Verlang, Berlin, Germany, 1995, Vol. 949, pp. 1-18.

[6] D.G. Feitelson, and L. Rudolph, "Coscheduling Based on Runtime Identification of Activity Working Sets", *International Journal of Parallel Programming*, Kluwer/Plenum, New York, USA, 1995, Vol. 23 (2), pp. 135-160.

[7] D. Jiang, and J. Pal Singh, "Scaling Application Performance on Cache-Coherent Multiprocessors", *Performance Evaluation Review*, ACM, New York, USA, 1998, Vol. 26 (1), pp. 171-181.

[8] H.D. Karatza, "Gang Scheduling and I/O Scheduling in a Multiprocessor System", In *Proceedings of 2000 Symposium on Performance Evaluation of Computer and Telecommunication Systems*, M.S. Obaidat, F. Davoli and M.A. Marsan (eds.), SCS, Vancouver, Canada, July 2000, pp. 245-252.

[9] H.D. Karatza, "A Simulation Based Performance Analysis of Scheduling in a Parallel System", In *Proceedings of 12th European Simulation Symposium and Exhibition*, SCS Europe, Hambourg, Germany, September 2000, pp. 582-586.

[10] H.D. Karatza, "Scheduling Jobs with Different Characteristics in a Partitionable Parallel System", In *Proceedings of the UKSim 2001 Conference*, UK Simulation Society, Cambridge, England, March 28-30, 2001, pp. 223-229.

[11] H.D. Karatza, and I.D. Scherson, "Scheduling Job Classes in a Distributed System". In *Proceedings of SPECTS'2001, 2001 SCS Symposium on Performance Evaluation of Computer and Telecommunication Systems*, SCS, Orlando, Florida, July 2001, pp. 322-329.

[12] Law, A., and D. Kelton, *Simulation Modeling and Analysis*, 2nd Ed., McGraw-Hill, Inc, New York, USA, 1991.

[13] S.K. Setia, "Trace-Driven Analysis of Migration-Based Gang Scheduling Policies for Parallel Computers", In *Proceedings of the International Conference on Parallel Processing*, IEEE Computer Society, Bloomingdale, USA, August 1997, pp. 489-492.

[14] F. Silva, and I.D. Scherson, "Improving Throughput and Utilization in Parallel Machines Through Concurrent Gang", In *Proceedings of the IEEE International Parallel and Distributed Processing Symposium 2000*, IEEE Computer Society, Cancun, Mexico, May 2000, pp. 121-126.

[15] F. Wang, M. Papaefthymiou, and M. Squillante, "Performance Evaluation of Gang Scheduling for Parallel and Distributed Systems", In *Job Scheduling Strategies for Parallel Processing*, *Lecture Notes in Computer Science*, D.G. Feitelson and L. Rudolph (eds.), Springer-Verlang, Berlin, Germany, 1997, Vol. 1291, pp. 184-195.