SCHEDULING PARALLEL AND SEQUENTIAL JOBS IN A PARTITIONABLE PARALLEL SYSTEM

HELEN D. KARATZA

Department of Informatics
Aristotle University of Thessaloniki
Email: karatza@csd.auth.gr

Abstract: This paper addresses performance issues associated with job scheduling in a partitionable parallel system. Jobs are characterized as sequential or parallel depending on whether their tasks have to be processed sequentially on the same processor or at different processors. Jobs that consist of parallel tasks 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 high system performance and maintain fairness in terms of sequential and parallel job execution. The performance of different scheduling schemes is compared over various workloads. Simulation results demonstrate that sequential jobs should not arbitrarily overtake the execution of parallel jobs.

***Keywords*:** Parallel Systems, Scheduling, Simulation.

1 INTRODUCTION

The topic of job scheduling policies for partitionable parallel systems has received considerable attention in recent years. The allocation and management of resources in these systems is fundamental to sustaining and improving the benefits of multiprocessing [Dowdy et al, 1999], [Karatza, 2000b].

Jobs usually have different characteristics. For example, there are jobs which consist of sequential tasks, while other jobs consist of parallel tasks that can be run in parallel on different processors. It is not always possible to efficiently execute all jobs. It is crucial to apply the proper strategy to jobs depending on their characteristics.

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

Our work considers a shared memory system with 128 processors. We study a scalable, coherent shared address space (SAS) multiprocessing system since it has been the focus of research in many other studies [Jiang and Singh, 1998], [Shan et al, 2000].

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 built and shown to perform well at the moderate scale of about 32 processors. In fact, such machines are fast becoming the dominant forms of tightly coupled multiprocessors built by commercial vendors.

An open question is how scalable these architecture configurations are to larger processor counts. Jiang and Singh (1998) studied the performance of a wide range of SAS parallel applications on a 128-processor hardware cache-coherent machine (the SGI Origin2000). They showed that scalable performance is indeed been 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.

Half of the total number of jobs are sequential while the rest are parallel jobs. Sequential jobs consist of tasks that run on the same processor.

Parallel jobs consist of parallel tasks that are scheduled to execute concurrently on a set of processors. The parallel tasks need to 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 has been extensively studied in the literature of parallel and distributed systems [Aida, 2000], [Aida et al, 1998], [Feitel-

son, 1994], [Feitelson and Jette, 1997], [Feitelson and Rudolph, 1995a], [Feitelson and Rudolph, 1995b], [Karatza, 2000a], [Karatza, 2000b], [Setia, 1997], [Silva and Scherson, 2000], [Wang et al, 1997].

Jobs start to execute only if enough idle processors are available to handle them. However, a scheduling policy is needed to determine which program is to be mapped to the available processors. Job sequencing needs to be preserved as much as possible in order to achieve fairness in job execution.

Generally, other papers found in the literature study processor scheduling only. They do not explicitly model the I/O processing, even though it can significantly influence the overall system performance. However, scheduling is not an isolated issue. It is only a single service provided by the operating system. Any solution to the scheduling problem must be integrated with other problem solutions, e.g. I/O management. Different parts of the system must work together to create a cohesive whole in such a way that it makes sense.
The study by Rosti et. al (1998) of large-scale parallel computer systems suggests that the overlapping of I/O demands of some jobs with the computation demands of other jobs offer a potential improvement in performance.

The design choices considered in this paper include different ways to schedule sequential jobs and gangs for service on the system processors.

The scheduling of sequential jobs and gangs in partitionable parallel systems has been studied in [Karatza, 2001], and [Karatza and Hilzer, 2002]. However those works do not examine the impact of the variability of task service demand on performance. Furthermore, the system that is studied in [Karatza and Hilzer, 2002] is different from the system that is studied in [Karatza, 2001] and in this work. That is, in [Karatza and Hilzer, 2002] the queuing network model is open, while in [Karatza, 2001] and in this paper closed queuing networks are considered.

In this paper we study and compare the performance of different scheduling policies for different degrees of multiprogramming and coefficients of variation of processor service times.

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 required 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 the subtleties 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 rather than 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 paper is structured 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 the performance of the scheduling policies.

Model implementation and input parameters are described in section 3.1, while the results of the simulation experiments are presented and analyzed in section 3.2.

Section 4 provides the conclusion and suggestions for further research; the last section includes the relevant references.

2 MODEL AND METHODOLOGY

2.1 System and Workload Models

A closed queuing network model is considered that consists of $P = 128$ parallel homogeneous processors and the I/O subsystem.

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 I/O subsystem may consist of an array of disks, but it is modeled as a single I/O node with a given mean service time.

Since we are interested in a system with balanced program flow, we consider an I/O subsystem with the same service capacity as the processing unit.

The model is considered closed since the degree of multiprogramming N is constant. The configuration of the model is shown in Figure 1, where m and k represent the mean processor and the mean I/O service time respectively.

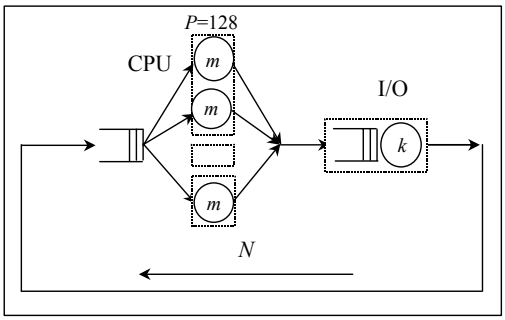

Figure 1: The Queuing Network Model

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 related to the:

- Number of job tasks.
- Interdependence of job tasks.
- Number of system jobs (degree of multiprogramming) that represent the system load.
- Variability in task service demand.

2.1.1 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 limit 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.

2.1.2 Interdependence among job tasks

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 $t_x \ge p(x)$.

Gangs
Those jobs that consist 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 is not currently running. Without gang scheduling, tasks will block in order to synchronize, thus suffering 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 $x_1, x_2, \ldots x_n$ can be executed simultaneously with s sequential jobs, where $0 \le s < P$, if and only if the following relation holds:

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

In our model, jobs that consist of $1 \le n \le 8$ tasks are sequential, while jobs that consist of $16 \le n \le 132$ tasks are gangs. Therefore, for every job there is a probability of 0.5 to be sequential, and 0.5 to be a gang.

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

After processor service, jobs request service from the I/O subsystem. The I/O queuing discipline is FCFS. Each time a job returns from I/O service to the processor system, it has a different number of tasks with different interdependence characteristics than the last time through the processors. That is, the degree of parallelism is not constant during a job's lifetime in the system.

2.1.3 Distribution of task service demand

We examine the impact of the variability in task service demand on system performance.

A high variability in task service demand implies that there is a proportionally high number of service demands that are very small compared to the mean service time and there is a comparatively small number of service demands that are very large. Tasks with a large service demand may introduce large queuing delays for queued jobs.

The parameter that represents the variability in task service demand is the coefficient of variation of task service demand C. We examine the following cases with regard to task service demand distribution:

- Task service demand is an exponentially distributed random variable with mean m. In this case $C = 1$.
- Task service demand has a Branching Erlang distribution with two stages. The coefficient of variation is $C > 1$ and the mean is m.

2.1.4 Distribution of I/O service time

The I/O service times are exponentially distributed with mean k.

Next we describe the scheduling strategies employed in this work. As with most studies we assume that scheduling overhead is negligible. We 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 Scheduling Strategies

Adapted First Come First Served (AFCFS)

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

AFCFS-Blocking of Sequential Jobs (AFCFS-BS)

When a job leaves the processors, 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 the processor queue in increasing job size order (larger gangs are placed ahead in the queue).

On the other hand it is well known that in cases where coscheduling is not required for tasks of jobs, then the shortest service time first is the optimal method. However, in most cases a priori knowledge of task service time is not available. For this reason, in this work 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 blocking of sequential jobs takes place in a similar manner as in the AFCFS –BS case.

2.3 Performance Metrics

We consider the following definitions:

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

Cycle time of a job is the elapsed time between two successive service requests of a job on the processors. This includes processor queuing and service times, plus I/O queuing and service times. The parameters used in simulation computations (presented later) are listed in Table 1.

Table 1: Notations

RT	Mean response time
K	Mean cycle time
R	System throughput
R_s	Throughput of sequential jobs
R_g	Throughput of gangs
D_R, D_{Rs}, D_{Rg}	Relative % increase in R, R_s, and R_g due to sequential jobs blocking
RT_S	Mean response time of sequential jobs
RT_G	Mean response time of gangs
U_{proc}	Mean processor utilization
N	Degree of multiprogramming
m	Mean processor service time
k	Mean I/O service time
C	Coefficient of variation of task service demand

3 SIMULATION RESULTS AND DISCUSSION

3.1 Model Implementation and Input Parameters

The queuing network model was simulated using discrete event simulation models using the independent replication method.

A balanced system with m=1.0 and k = 0.249 was considered. The value k=0.249 was chosen for balanced program flow 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 is served during each time unit. This implies that I/O mean service time must be equal to 1 / 4.0157 = 0.249 if the I/O unit is to have the same service capacity.

The system is examined for cases of task execution times following exponential (C=1) and Branching Erlang (C=2) distributions, respectively. N is either 64, 80, 96, 112, or 128. The reason for examining different degrees of multiprogramming is that it is a critical driver of system load.

3.2 Performance Analysis

The results that follow next are representative of the system and program models.

- Tables 2-5 present performance parameters for the C = 2 case.
- Figures 2-3 show D_R and RT ratio versus N for C = 1 in the cases where AFCS-BS and LG-SS-BS policies are compared with the AFCFS and LG-SS methods, respectively.
- Figures 4-5 present D_{Rs} and D_{Rg} versus N for C = 1 comparing the AFCFS and LG-SS cases, respectively.
- Figures 6-7 present R_s and R_g versus N for C = 1 comparing the AFCFS, AFCFS-BS, and LG-SS, LG-SS-BS cases, respectively.
- Figures 8-9 present RT_s and RT_g versus N for C = 1 comparing the AFCFS, AFCFS-BS, and LG-SS, LG-SS-BS cases, respectively.
- Figures 10-11 present D_R and RT ratio versus N for C = 2 comparing the AFCFS, AFCFS-BS, and LG-SS, LG-SS-BS cases, respectively.
- Figures 12-13 present D_{Rs} and D_{Rg} versus N for C = 2 comparing the AFCFS and LG-SS cases, respectively.
- Figures 14-15 present R_s and R_g versus N for C = 2 comparing the AFCFS, AFCFS-BS, and LG-SS, LG-SS-BS cases, respectively.
- Figures 16-17 present RT_s and RT_g versus N for C = 2 comparing the AFCFS, AFCFS-BS, and LG-SS, LG-SS-BS cases, respectively.

Table 2: AFCFS policy, C = 2

N	U_{proc}	RT	K	R
64	0.62	21.98	26.00	2.46
80	0.64	25.86	31.22	2.56
96	0.65	29.95	36.70	2.62
112	0.67	33.76	42.10	2.66
128	0.67	37.64	47.43	2.70

Table 3: AFCFS-BS policy, C = 2

N	U_{proc}	RT	K	R
64	0.63	21.49	25.33	2.53
80	0.68	24.01	29.23	2.74
96	0.72	26.88	33.48	2.87
112	0.75	29.27	37.24	3.01
128	0.77	32.34	41.65	3.07

Table 4: LG-SS policy, $C = 2$

N	U_{proc}	RT	K	R
64	0.62	21.04	25.72	2.49
80	0.64	25.22	31.38	2.55
96	0.65	28.81	36.64	2.62
112	0.66	33.13	42.36	2.64
128	0.67	37.03	47.83	2.68

Table 5: LG-SS-BS policy, $C = 2$

N	U_{proc}	RT	K	R
64	0.66	19.00	24.10	2.66
80	0.70	21.90	28.69	2.79
96	0.72	24.95	33.26	2.89
112	0.75	27.79	37.65	2.98
128	0.76	30.96	42.36	3.02

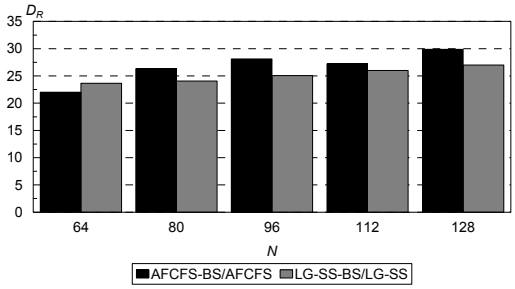

Figure 2: D_R versus N, $C = 1$

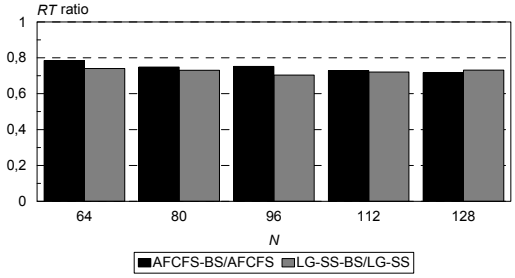

Figure 3: Response time ratio versus N, $C = 1$

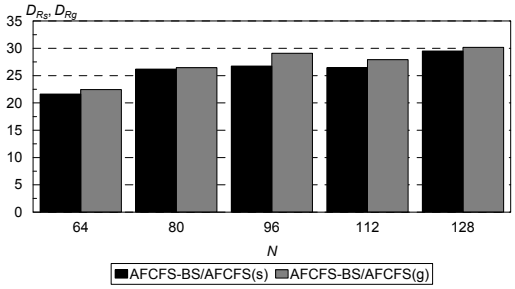

Figure 4: D_{Rs} and D_{Rg} versus N for the AFCFS case, $C = 1$

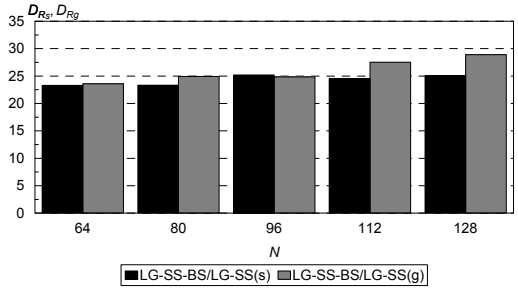

Figure 5: D_{Rs} and D_{Rg} versus N for the LG-SS case, $C = 1$

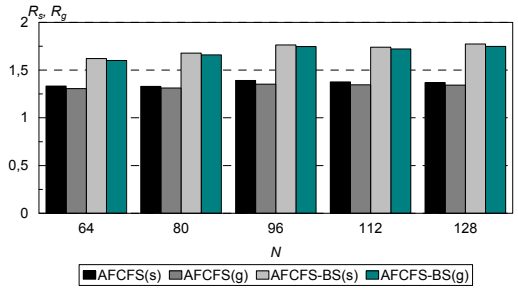

Figure 6: R_s and R_g versus N for the AFCFS and AFCFS-BS cases, $C = 1$

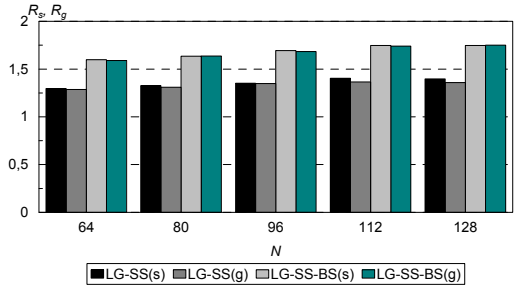

Figure 7: R_s and R_g versus N for the LG-SS and LG-SS-BS cases, $C = 1$

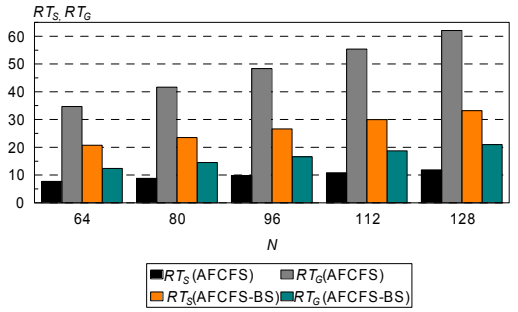

Figure 8: RT_S and RT_G versus N for the AFCFS and AFCFS-BS cases, $C = 1$

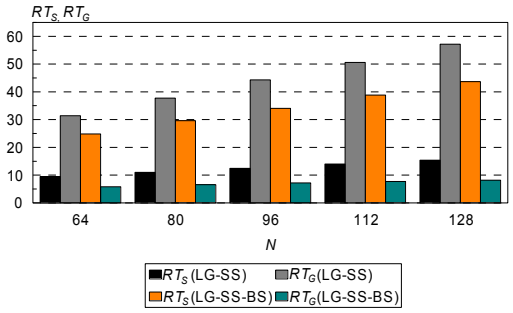

Figure 9: RT_S and RT_G versus N for the LGFS-SS and LGFS-SS-BS cases, $C = 1$

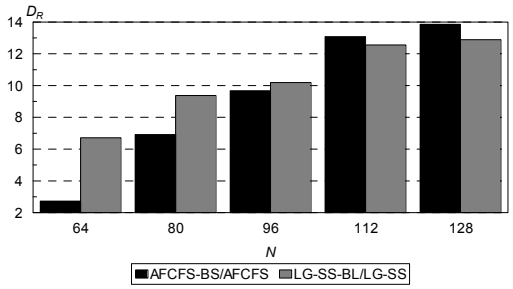

Figure 10: D_R versus N, $C = 2$

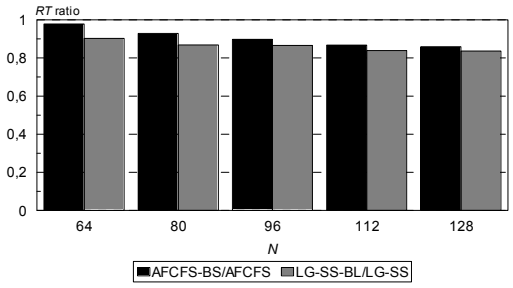

Figure 11: Response time ratio versus N, $C = 2$

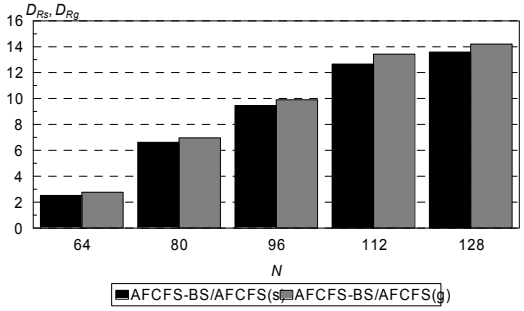

Figure 12: D_{Rs} and D_{Rg} versus N for the AFCFS case, $C = 2$

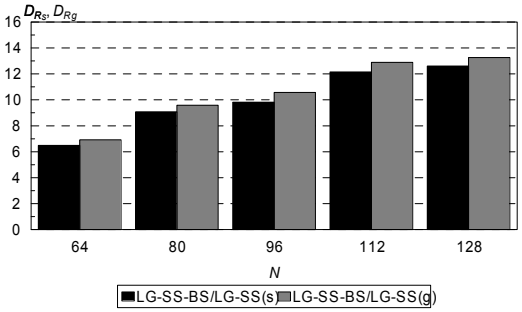

Figure 13: D_{Rs} and D_{Rg} versus N for the LG-SS case, $C = 2$

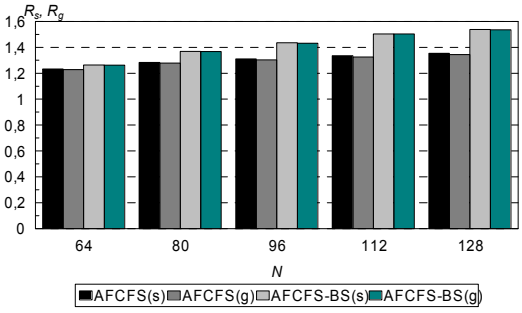

Figure 14: R_s and R_g versus N for the AFCFS and AFCFS-BS cases, $C = 2$

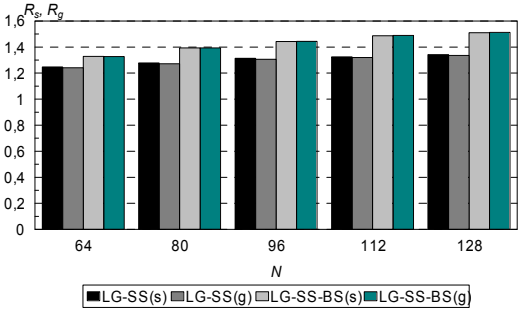

Figure 15: R_s and R_g versus N for the LG-SS and LG-SS-BS cases, $C = 2$

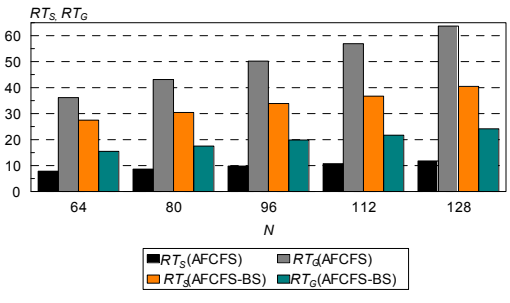

Figure 16: RT_S and RT_G versus N for the AFCFS and AFCFS-BS cases, $C = 2$

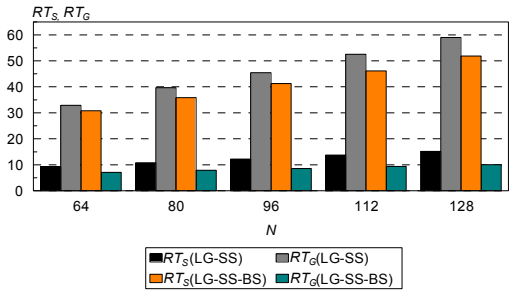

Figure 17: RT_S and RT_G versus N for the LGFS-SS and LGFS-SS-BS cases, $C = 2$

The results demonstrate the following:

As far as overall performance is concerned, the AFCFS-BS and LG-SS-BS methods perform better than the AFCFS and LG-SS methods respectively (Tables 2-5, Figures 2 and 10). This is due to the fact that the mean response time is lower in the AFCFS-BS and LG-SS-BS cases than in the AFCFS and LG-SS cases, respectively. This results in lower cycle time, higher throughput, and higher utilization of system units.

Therefore, the blocking of sequential jobs improves the overall 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 period 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. Consequently, in the non blocking cases gangs are accumulated in the queue and suffer long delays. These delays of gangs in the non blocking cases affect the mean waiting time of all jobs more seriously than the delays of sequential jobs affect in the blocking cases. Also, a consequence of long delays of gangs might be starvation of the I/O subsystem.

Generally, the performance of the AFCFS-BS and the LG-SS-BS polices does not differ significantly. This is shown in Tables 3 and 5 for the $C = 2$ case. In some cases AFCFS-BS performs slightly better than LG-SS-BS, while in some other cases LG-SS-BS performs slightly better than AFCFS-BS. We have to notice though that the AFCFS-BS method is of easier to implementation and therefore causes less overhead than the LG-SS-BS method. In each job class, job sequencing is better preserved with the AFCFS-BS policy than with the LG-SS-BS method. Therefore, from these two methods the fairest method is AFCFS-BS.

In Figures 2 and 10 we observe the relative percentage increase in throughput due to sequential jobs blocking for $C = 1$ and $C = 2$, respectively. The increase is larger in the $C = 1$ case than in the $C = 2$ case. It varies between 22% and 30% in the $C = 1$ case and between 2.8% and 14% in the $C = 2$ case. The results presented in these two Figures show that in most cases sequential jobs blocking improves overall system performance more significantly at large degrees of multiprogramming.

Figures 3 ($C = 1$) and 11 ($C = 2$) presents the response time ratio, which is the ratio of RT of a policy that blocks sequential jobs, divided by the RT for the corresponding non-blocking policy. This ratio varies approximately in the ranges of [0.7, 0.79] and [0.84, 0.98] in the $C = 1$ and $C = 2$ cases respectively.

In most cases the response time ratio for the AFCFS-BS policy is slightly larger than in the LG-SS-BS case. This means that typically the blocking of sequential jobs affects the mean response time of all jobs more significantly in the LG-SS-BS case than in AFCFS-BS case. This is due to the fact that when a sequential job is blocked, a larger gang may be scheduled in the LG-SS-BS case than in the AFCFS-BS. This results in a more efficient usage of the available processors and in a smaller mean response time. In the remaining cases, the response time ratio for the AFCFS-BS policy does not differ significantly from the ratio in the LG-SS-BS case.

Regarding the response time of sequential jobs and of gangs, in all cases blocking of sequential jobs increases RT_s and decreases RT_g (Figures 8, 9 for the $C = 1$ case, and Figures 16, 17 for the $C = 2$ case).

In Figures 4, 5, 6, and 7 ($C = 1$) and in Figures 12, 13, 14, and 15 ($C = 2$), we observe that not only the throughput rate of gangs improves with the BS versions of the AFCFS and LG-SS strategies, but also the throughput rate of sequential jobs improves. This is explained in the following.

When a sequential job is blocked, then the waiting time of this job is increased. However, blocking may result in a more efficient usage of the available processors, by assigning them to a gang that is waiting in the queue. A gang keeps its assigned processors busy for a time period that is on average much smaller than the time period required for the service of the sequential job. There-

fore, all these processors can earlier serve more subsequent jobs than in the case of priority allocation to individual sequential jobs.

The results have been obtained without taking into account the overhead that depends on the complexity of the scheduling policy that is employed. However, we have mentioned already that LG-SS-BS involves extra overhead as compared to AFCFS-BS.

4 CONCLUSIONS AND FURTHER RESEARCH

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

Four scheduling policies were considered (AFCFS, LG-SS, AFCFS-BS and LG-SS-BS). Their performance was simulated and then compared for various degrees of multiprogramming and for different coefficients of variation of processor service times. The simulation results reveal the following:

- Blocking of sequential jobs decreases mean response time of gangs but increases mean response time of sequential jobs. However, it improves overall throughput, and also improves throughput of sequential jobs and throughput of gangs.

- Between the AFCFS-BS and LG-SS-BS policies, the first method should be used as it involves less overhead than LG-SS-BS and because in most cases the two methods do not differ significantly. Furthermore, job sequencing within each job class is better preserved with the AFCFS-BS method than with the LG-SS-BS method.

- System and program performance depend on the workload.

This research can be extended to consider parallel jobs that consist of independent tasks that can be executed at any processor and in any order, along with the sequential jobs and gangs. Also, different distributions for the I/O service times could be examined.

REFERENCES

Aida K. 2000, "Effect of Job Size Characteristics on Job Scheduling Performance". In *Proc. of the 6th Workshop on Job Scheduling Strategies for Parallel Processing. Lecture Notes in Computer Science*, ed. D.G. Feitelson and L. Rudolph, Springer-Verlang, Berlin, Germany. Vol. 1911. Pp1-10.

Aida K., Kasahara H. and Narita S. 1998, "Job Scheduling Scheme for Pure Space Sharing among Rigid Jobs". In *Job Scheduling Strategies for Parallel Processing*, *Lecture Notes in Computer Science*, ed. D.G. Feitelson and L. Rudolph, Springer-Verlang, Berlin, Germany. Vol. 1459. Pp98-121.

Dowdy L.W., Rosti E., Serazzi G. and Smirni E. 1999, "Scheduling Issues in High-Performance Computing". *Performance Evaluation Review*, ACM, New York, USA. Vol. 26 (4). Pp60-69.

Feitelson D.G. 1994, *A Survey of Scheduling in Multiprogrammed Parallel Systems*. Research Report RC 19790 (87657), IBM T.J. Watson Research Center, Yorktown Heights, New York, USA.

Feitelson D.G. and Jette M.A. 1997, "Improved Utilization and Responsiveness with Gang Scheduling". In *Job Scheduling Strategies for Parallel Processing*, *Lecture Notes in Computer Science*, ed. D.G. Feitelson and L. Rudolph, Springer-Verlang, Berlin, Germany. Vol. 1291. Pp238-261

Feitelson D.G. and Rudolph L. 1995a, "Parallel Job Scheduling: Issues and Approaches". In *Job Scheduling Strategies for Parallel Processing*, *Lecture Notes in Computer Science*, ed. D.G. Feitelson and L. Rudolph, Springer-Verlang, Berlin, Germany. Vol. 949. Pp1-18.

Feitelson D.G. and Rudolph L. 1995b, "Coscheduling Based on Runtime Identification of Activity Working Sets". *International Journal of Parallel Programming*. Kluwer/Plenum, New York, USA. Vol. 23 (2). Pp135-160.

Jiang D. and Pal Singh J. 1998, "A Methodology and an Evaluation of the SGI Origin2000". *Performance Evaluation Review*. ACM, New York. USA. Vol. 26 (1). Pp171-181.

Karatza H.D. 2000a, "Gang Scheduling and I/O Scheduling in a Multiprocessor System". In *Proc. of 2000 Symposium on Performance Evaluation*

of Computer and Telecommunication Systems, ed. M.S. Obaidat, F. Davoli, and M.A. Marsan (Vancouver, Canada, July) SCSI, San Diego, CA, USA. Pp245-252.

Karatza H.D. 2000b, "A Simulation Based Performance Analysis of Scheduling in a Parallel System". In *Proc. of 12th European Simulation Symposium and Exhibition* (Hambourg, Germany, September) SCS Europe, Ghent, Belgium. Pp582-586.

Karatza H.D. 2001, "Scheduling Jobs with Different Characteristics in a Partitionable Parallel System". In *Proc. of the UKSim 2001 Conference* (Cambridge, England, March) UK Simulation Society, Nottingham, England. Pp223-229.

Karatza H.D. and Hilzer R.C. 2002, "Scheduling a Job Mix in a Partitionable Parallel System". In *Proc. of the 35th Annual Simulation Symposium* (San Diego, California, April) IEEE Computer Society Press, Los Alamitos, CA, USA. Pp235-241.

Rosti E., Serazzi G., Smirni E. and Squillante M. 1998, "The Impact of I/O on Program Behavior and Parallel Scheduling". *Performance Evaluation Review*. ACM, New York, USA. Vol. 26 (1). Pp56-65.

Setia S.K. 1997, "Trace-Driven Analysis of Migration-Based Gang Scheduling Policies for Parallel Computers". In *Proc. of the International Conference on Parallel Processing* (Bloomingdale, USA, August) IEEE Computer Society, Los Alamitos, CA, USA. Pp489-492.

Shan H., Feng J. and Shan H. 2000, "Programming FFT on DSM Multiprocessors". In *Proc. of the Fourth International Conference/Exhibition on High Performance Computing in Asia-Pacific Region* (Beijin, China, May) IEEE Computer Society Press, Los Alamitos, CA, USA. Pp599-606.

Silva F. and Scherson I.D. 2000, "Improving Throughput and Utilization in Parallel Machines Through Concurrent Gang". In *Proc. of the IEEE International Parallel and Distributed Processing Symposium 2000* (Cancun, Mexico, May) IEEE Computer Society, Los Alamitos, CA, USA. Pp121-126.

Wang F., Papaefthymiou M. and Squillante M.S. 1997, "Performance Evaluation of Gang Scheduling for Parallel and Distributed Systems". In *Job Scheduling Strategies for Parallel Processing*, *Lecture Notes in Computer Science*, ed. D.G. Feitelson and L. Rudolph, Springer-Verlang, Berlin, Germany. Vol. 1291. Pp184-195.

BIOGRAPHY

HELEN D. KARATZA is an Associate Professor in the Department of Informatics at the Aristotle University of Thessaloniki, Greece. Her research interests mainly include Performance Evaluation of Parallel and Distributed Systems, Multiprocessor Scheduling, Mobile Agents, Mobile Computing, and Simulation. Dr. Karatza is a member of the Editorial Board of the International Journal of Simulation: Systems, Science & Technology (the UK Simulation Society), Associate Editor of the Journal Simulation: Transactions of the Society for Modeling and Simulation International (Applications Section), and area Editor for computer systems of the Journal of Systems and Software (Elsevier). She has served as a member of Program Committees and Program Chair of many Simulation related International Conferences/Symposia. Her email and web address are <karatza@csd.auth.gr>, and <agent.csd.auth.gr/~karatza>.