PERFORMANCE ANALYSIS OF GANG SCHEDULING IN A PARTITIONABLE PARALLEL SYSTEM

Helen D. Karatza
Department of Informatics
Aristotle University of Thessaloniki
54124 Thessaloniki, Greece
E-mail: karatza@csd.auth.gr

KEYWORDS
Performance Analysis, Gang Scheduling.

ABSTRACT

This paper addresses performance issues associated with job scheduling in a partitionable parallel system. Jobs consist of parallel tasks scheduled to execute concurrently on processor partitions, where each task starts at the same time and computes at the same pace. The performance of different scheduling schemes is compared over various workloads. Various performance metrics are examined. The objective is to achieve good overall performance and also small scheduling overhead. Simulated results indicate that periodic job scheduling and also scheduling which depends on the number of job insertions in the queue can succeed in these pursuits.

INTRODUCTION

The efficient scheduling of parallel jobs on processors of multiprogrammed parallel systems is critical to achieving high performance. Users expect their individual jobs to achieve excellent performance. The main issue is how to share system resources among competing jobs, in a way that satisfies the demands of jobs and produces good overall performance. These objectives raise a number of scheduling policy issues with respect to large parallel computing environments.

This study considers a partitionable parallel system where the partitions are subsystems allocated to independent jobs (Li, 1997). 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 distributed and shared memory systems (Aida 2000; Corbalan et al. 2001; Feitelson and Jette 1997; Frachtenberg et al. 2005; Karatza 2001a; Karatza 2001b; Karatza 2002; Setia 1997; Strazdins and Uhlmann 2004; Wang et al. 1997; Wiseman and Feitelson 2003; Zhang et al. 2003a; Zhang et al. 2003b). Common reasons to use gang scheduling are its responsiveness and efficient use of resources. Some examples of gang scheduling use are its implementation on the CM-5 Connection Machine, IBM SP2 and clusters of workstations.

Jobs start to execute only if enough idle processors are available to handle them. However, a scheduling policy is needed to determine which parallel program is to be mapped to the available processors. In multiprogrammed parallel systems, processor partitions are usually allocated on a First Come First Served (FCFS) basis. This approach can result in severe fragmentation, because processors that cannot fulfil demands of the next job in the queue remain idle until the needed resources are freed. To avoid fragmentation, a non-FCFS policy for queuing waiting jobs on a partitionable system should be used.

In (Karatza 2001a) we studied the performance of two well-known gang scheduling methods, the Largest Job First Served (LJFS) and the Adapted First Come First Served (AFCFS). This paper considers closed queuing network models with a fixed number of jobs. It has been shown that in many cases LJFS performs better than AFCFS. However, LJFS has the disadvantage that it involves a considerable amount of overhead because the processor queue is re-arranged each time a new parallel job is added.

Most research into parallel job scheduling policies has focused on improving overall performance where scheduling overhead is assumed to be negligible. However, scheduling overhead can seriously degrade performance. In this work, along with the AFCFS and LJFS scheduling methods, we also consider two other policies: the Periodic Largest Job First Served (PLJFS) and the Queue insertions dependent LJFS (QLJFS) scheduling methods. With the PLJFS policy the processors queue is re-arranged only at the end of predefined time periods p. At the end of a period the scheduler recalculates the priorities of all jobs in the queue using the LJFS criterion. When the QLJFS policy is employed, the queue is re-arranged according to the LGFS criterion every i job insertions in the queue.

We aim to find if the periodic and the queue insertions dependent scheduling methods perform well as compared to the LJFS policy and minimize the disadvantage of LJFS as much as possible. Scheduling

Proceedings 20th European Conference on Modelling and Simulation
Wolfgang Borutzky, Alessandra Orsoni, Richard Zobel © ECMS, 2006
ISBN 0-9553018-0-7 / ISBN 0-9553018-1-5 (CD)

optimality is defined as minimizing the number of queue re-arrangements. We study and compare the scheduling policies for various workloads and for different periods p and numbers i of jobs insertions in the queue. Comparative results are obtained using simulation techniques.

In a previous paper (Karatza 2001b) we studied an epoch scheduling method. However, in that paper the system and workload models are different than those that are examined here. That paper studies a distributed system, where each processor is equipped with its own queue. It considers a closed queuing network model with a fixed number of jobs. Further to this, it does not examine gang scheduling. It considers jobs with independent tasks that can execute on any processor and in any order.

Periodic gang scheduling in an open queueing network model of a partitionable parallel system has been studied in (Karatza 2002). In that paper the overall performance is expressed by the average response time of jobs. In this paper we study additional metrics, which better reflect performance of scheduling strategies. One of them is the average slowdown. Furthermore, we examine two other metrics the average weighted response time and the average weighted slowdown where the weight is the number of processors required by a job, which is a job's degree of parallelism. Thereby, it is avoided that jobs with the same execution time, but with different resource requirements, have the same impact on the overall performance. Furthermore, this paper studies an additional scheduling method, the queue insertions dependent method that is not studied in (Karatza 2002). To our knowledge, gang scheduling in partitionable parallel systems operating under these workload models has not appeared in the research literature.

The paper consists of the following five sections: The first section specifies system and workload models, it describes the scheduling strategies, and it presents the metrics employed while assessing performance of the scheduling strategies. Experimental methodology is described in the second section, and experimental results are presented and analyzed in the third section. The fourth section contains conclusions and suggestions for further research, and the fifth section is the References.

MODEL AND METHODOLOGY

System and Workload Models

An open queuing network model is considered that consists of P = 128 parallel homogeneous processors (Figure 1). An example of a machine of this size is Sweetgum which is an SGI Origin 2800 Supercomputer equipped with 128 CPUs. 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.

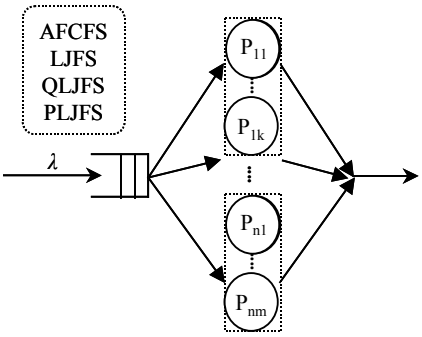

Figure 1: The Queuing Network Model, where λ is the Mean Job Inter-Arrival Time

A partitionable parallel processing system is used which dynamically allocates jobs to processor subsystems. 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 processors required by job x is represented as $p(x)$, and is called the "size" of job x. A job is said to be "small" (or "large") if it requires a small (or large) number of processors. It is obvious that $t_x = p(x)$. The $p(x)$ processors must be allocated simultaneously to job x, and once they are allocated, they are held by job x until its completion. Jobs $x_1, x_2, ... x_j$ can be executed simultaneously if and only if the following relation holds: $p(x_1)+p(x_2)+...+p(x_j) \le P$.

Each job begins execution only when enough idle processors is available to meet its needs. When a job terminates execution, all processors assigned to it are reclaimed. We assume that there is no correlation between job size and task service demand. For example, a small job may have a long service time. The number of jobs that can be processed in parallel depends on job sizes and on the scheduling policy applied.

We evaluate the performance of job scheduling algorithms under various workload models, each of which has certain characteristics relating to: a) The distribution of the number of job tasks, b) The distribution of job inter-arrival time, and c) The distribution of task service demand.

Distribution of job sizes: We assume that job sizes are uniformly distributed over the range [1..128].

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 are exponentially distributed with a mean of $1/\mu$.

Job Scheduling Policies

Adapted First Come First Served (AFCFS). This method schedules jobs whenever enough processors are available. When the number of available processors is not sufficient for a large job that is waiting at the head of the ready queue, then the AFCFS policy considers the scheduling of smaller jobs ahead of the large job. One major problem with this scheduling policy is that it tends to favour those jobs requesting a smaller number of processors and thus may increase the fragmentation in the system.

Largest Job First Served (LJFS). With this policy jobs are placed in increasing job size order in the processor queue (jobs that consist of a large number of parallel tasks are placed ahead in the queue). All jobs in the queue are searched in order, and the first jobs for which the assigned processors are available start 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, supercomputer centers have a mandate to run large, highly parallel jobs that cannot run anywhere else.

Queue insertions dependent Largest Job First Served (QLJFS). With this policy arriving jobs are placed at the end of the queue. The queue is re-arranged according to the LJFS criterion every i job insertions in the queue. That is, the queue is re-arranged only at the times where the number of the total number of queue insertions q is a multiple of i, i.e. when $q \bmod i = 0$.

Periodic Largest Job First Served (PLJFS). With this policy the processors queue is re-arranged only at the end of predefined time periods p. Then the scheduler recalculates the priorities of all parallel jobs in the queue using the LJFS criterion.

The goal of QLJFS and PLJFS is to decrease the number of queue re-arrangements as compared to LJFS, and to provide good performance.

Performance Metrics

Response time r_j of a job j 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).

Another parameter is the *slowdown* metric. The slowdown of a job is the job's response time divided by the job's execution (run) time. Additionally, we weight each job's response time and slowdown with its size (Streit, 2004). Thereby, it is avoided that jobs with the same execution time, but with different number of parallel tasks, have the same impact on the overall performance. For this reason, the *weighted response time* and *weighted slowdown* are examined. Parameters used in simulation computations (presented later) are shown in Table 1.

Table 1: Notations

P	Number of processors
μ	Mean processor service rate
$1/\mu$	Mean processor service time
λ	Mean job arrival rate
$1/\lambda$	Mean job inter-arrival time
i	The LJFS policy is employed every i job insertions in the queue
p	Period size
RT	Average response time
D_{RT}	Relative (%) decrease in RT when one of the LJFS, QLJFS, PLJFS methods are employed instead of the AFCFS policy
WRT	Average weighted response time
D_{WRT}	Relative (%) decrease in WRT when one of the LJFS, QLJFS, PLJFS methods are employed instead of the AFCFS policy
SLD	Average slowdown
D_{SLD}	Relative (%) decrease in SLD when one of the LJFS, QLJFS, PLJFS methods are employed instead of the AFCFS policy
$WSLD$	Average weighted slowdown
D_{WSLD}	Relative (%) decrease in $WSLD$ when one of the LJFS, QLJFS, PLJFS methods are employed instead of the AFCFS policy
NQR	Number of Queue Re-arrangements
D_{NQR}	Relative (%) decrease in NQR when QLJFS or PLJFS is employed instead of the LJFS policy

Let's l is the total number of processed jobs. If e_j is the execution time (service time) of job x_j, $j = 1, 2,.. l$, then the slowdown s_j of job x_j, is defined as follows:

$$s_l = r_j / e_j$$

The following metrics used for performance evaluation are defined as follows (Streit, 2004):

- The average response time RT:

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

- The average weighted response time WRT:

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

- The average slowdown SLD:

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

- The average weighted slowdown *WSLD*:

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

EXPERIMENTAL METHODOLOGY

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

We have chosen mean processor service time $1/\mu$ = 1, which means mean service rate per processor μ = 1. In the simulation experiments we set: $1/\lambda$ = 0.55, 0.60, and 0.65, which corresponds respectively to arrival rate: λ = 1.818, 1.667, and 1.538.

The value $1/\lambda$ = 0.55 is chosen as starting point for the experiments because the processors average (P+1) / 2 = 64.5 tasks per parallel job. Therefore, when all processors are busy an average of P / 64.5 = 1.9845 parallel jobs can be served each unit of time. This implies that we have to set λ < 1.9845, and consequently $1/\lambda$ > 0.504, i.e. the processors queues will not be saturated. However, due to gang scheduling there are often idle processors although there are jobs in the queue. Therefore the queue gets very easily saturated when mean inter-arrival time is close to 0.504. After experimental runs with various values of $1/\lambda$ we chose 0.55 as the smallest mean inter-arrival time for the experiments. Smaller mean interarrival times resulted in a sharp increase in average response time.

In the QLJFS case we examined two cases for the number of job insertions in the queue: i = 5, and 10.

In the PLJFS case we examined periods p = 4, 8, and 12. We chose period length 4 as a starting point for the experiments because the mean processor service time is equal to 1, and also because with this period size *NQR* is much smaller than in the LJFS case. Therefore we expected that larger period sizes would result in even smaller *NQR*.

EXPERIMENTAL RESULTS AND DISCUSSION

The results that follow represent the performance of the different policies. With all scheduling methods, system load (mean processor utilization) is 0.92, 0.84, 0.78 for $1/\lambda$ = 0.55, 0.60, 0.65 respectively.

We have to refer though that due to gang scheduling there are cases where some processors are kept idle although there are jobs waiting in the queue.

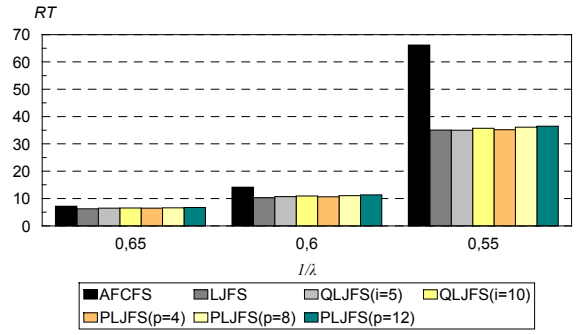

Figure 2: RT versus $1/\lambda$

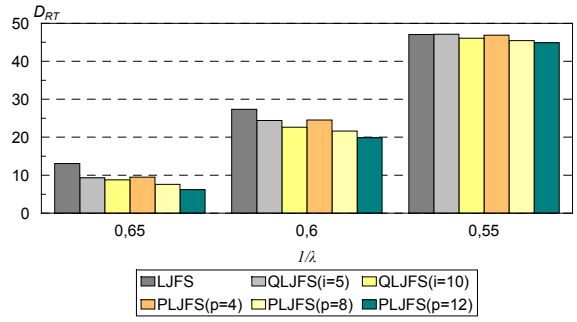

Figure 3: D_{RT} versus $1/\lambda$

In Figures 2 and 3 we observe that with regard to average response time the worst performance appears with the AFCFS policy because this method yields the highest average response time. In the QLJFS case the average response time is smaller when i = 5 than when i = 10. In the PLJFS case average response time increases with increasing period size. For $1/\lambda$ = 0.65, 0.60 the lowest average response time is produced by the LJFS policy and therefore this method yields the best overall performance. In these cases QLJFS(i=5) and PSJFS(p=4) perform almost the same. The difference between LJFS and each of QLJFS(i=5) and PSJFS(p=4) is larger in the $1/\lambda$ = 0.65 case than in the case of $1/\lambda$ = 0.60. For $1/\lambda$ = 0.55 LJFS performs almost the same as QLJFS(i=5) and PSJFS(p=4). In all three load cases QLJFS(i=10) performs better than PLJFS(p=8) and PLJFS(p=12).

In Figure 3 we also observe that D_{RT} increases with increasing load. This is because there are fewer jobs in the queue when $1/\lambda$ is large than when it is small. Therefore, there are fewer opportunities to exploit the advantages of the LJFS, QLJFS, and PLJFS methods over the AFCFS method when the load is small than when it is large.

Figure 4 shows that with all scheduling methods average weighted response times are larger than average response times. In Figure 5 it is shown that the

observations that hold for the relative performance of the scheduling policies with regard to average response time, also hold with regard to the average weighted response time. D_{WRT} is larger than D_{RT}. Therefore, the superiority of scheduling methods over AFCFS appears more significant when job response time is weighted by job degree of job parallelism.

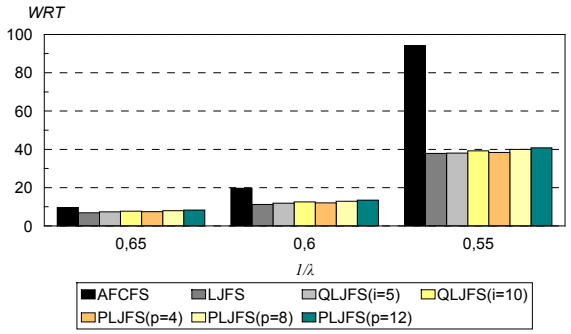

Figure 4: WRT versus $1/\lambda$

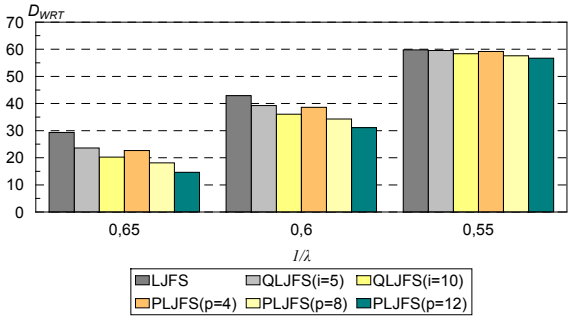

Figure 5: D_{WRT} versus $1/\lambda$

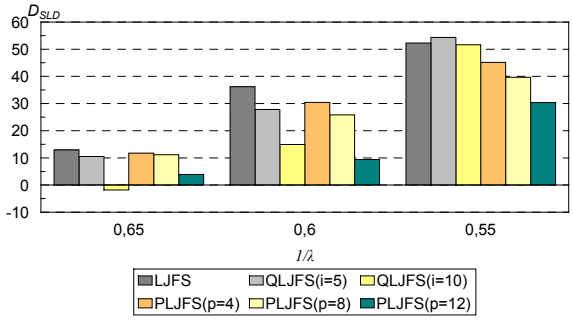

Figure 6: D_{SLD} versus $1/\lambda$

In Figure 6 we observe performance with regard to average slowdown. The worst performance appears with the AFCFS policy with one exception only in the $1/\lambda = 0.65$ case where the QLJFS(i=10) method yields slightly larger average slowdown as compared to AFCFS. With the QLJFS method the average slowdown is smaller when $i = 5$ than when $i = 10$. In the PLJFS case average slowdown increases with increasing period size. For $1/\lambda = 0.65$, 0.60 the lowest average slowdown is produced by the LJFS policy and therefore this method yields the best overall performance. In these cases PSLFS(p=4) performs better than QLJFS(i=5). In the $1/\lambda = 0.65$ case the PLJFS(p=8) method performs close to PLJFS(p=4) and QLJFS(i=5). For $1/\lambda = 0.60$, PLJFS(p=8) performs worse than QLJFS(i=5), but it performs better than QLJFS (i=10). The difference between LJFS and each of QLJFS and PLJFS is larger in the $1/\lambda = 0.60$ case than in the case of $1/\lambda = 0.65$. For $1/\lambda = 0.55$ QLJFS(i=5) is the best method. LJFS performs almost the same as QLJFS(i=10), whereas QLJFS(i=10) performs better than PLJFS (for all p). Figure 6 also shows that D_{SLD} increases with increasing load due to reasons explained in the analysis of D_{RT}.

Figure 7 shows that with regard to average weighted slowdown in all cases AFCFS is the worst method. For $1/\lambda = 0.55$, QLJFS(i=5) is the best method. LJFS performs better than QLJFS(i=10), whereas QLJFS(i=10) performs better than PLJFS (for all p). For $1/\lambda = 0.65$, 0.60, LGFS is the best method. QLJFS(i=5) performs better than PLJFS for all p, whereas PLJFS for $p = 4$, 8 performs better than QLJFS(i=10). D_{WSLD} is larger than D_{SLD}. In Figure 7 we also observe that D_{WSLD} increases with increasing load due to reasons explained in the analysis of D_{RT}.

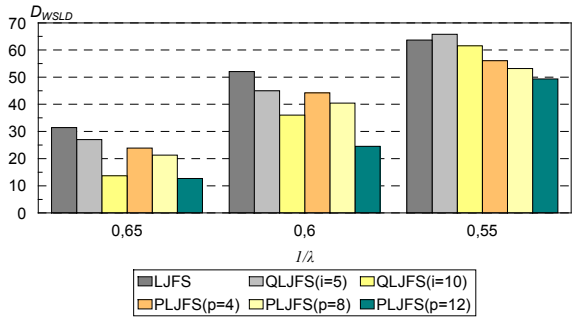

Figure 7: D_{WSLD} versus $1/\lambda$

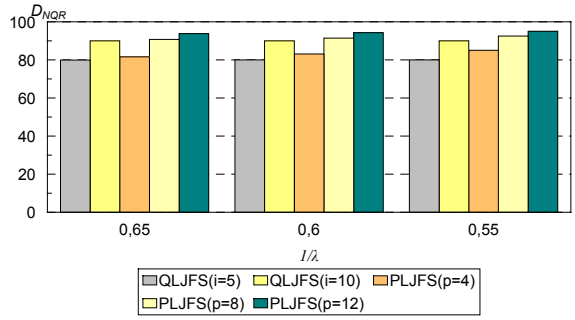

Figure 8: D_{NQR} versus $1/\lambda$

With regard to scheduling overhead the results presented in Figure 8 reveal that for all λ, the relative decrease in the number of queue re-arrangements due to QLJFS and PLJFS scheduling is very high. In all cases D_{NQR} varies in the range of 80 - 95%. In the QLJFS case D_{NQR} is larger when i=10 than when i=5. In the PLJFS case D_{NQR} increases with increasing period size. D_{NQR} increase is larger when period size changes from 4 to 8, than when changes from 8 to 12.

With each of QLJFS and PLJFS, D_{NQR} is almost the same in the three system load cases. Therefore, when QLJFS or PLJFS are employed instead of the LJFS method, the relative decrease in scheduling overhead is almost the same in the three different cases of system load. From all methods, the best is PLJFS(p=12). PLJFS(p=8) is slightly better than QLJFS(i=10), and PLJFS(p=4) is slightly better than QLJFS(i=5).

CONCLUSIONS AND FURTHER RESEARCH

This paper studies gang scheduling in a partitionable parallel system. Simulation is used to generate results needed to compare the performance of different scheduling policies under various workload models.

Along with two traditional scheduling methods: the AFCFS and the LJFS, also Queue dependent scheduling (QLJFS) and Periodic scheduling (PLJFS) are studied. Simulation results show that the relative performance of the different scheduling methods depends on the workload. With regard to all performance metrics considered in this paper, the LJFS method either performs best or close to QLJFS and PLJFS for some i and p respectively. At high system load QLJFS for small i performs either slightly better than LJFS (with regard to *SLD* and *WSLD*) or almost the same as LJFS (with regard to *RT* and *WRT*). It is important to notice that QLJFS and PLJFS involve significantly less overhead than the LJFS policy.

Future work could expand the analysis presented in this paper. Further experimentation is required to examine the performance of the gang scheduling methods in cases where the service demands of parallel jobs present high variability.

REFERENCES

Aida, K. 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, Vol. 1911, 1-10.

Corbalan J.; X. Martorell; and J. Labarta. 2001. "Improving Gang Scheduling through Job Performance Analysis and Malleability". In *Proceedings of the 2001 International Conference on Supercomputing* (Sorrento, Napoli, Italy, Jun.16-21). ACM, New York, NY, 303-311.

Feitelson, D.G. and M. A. Jette. 1997. "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, Vol. 1291, 238-261.

Frachtenberg, E.; D.G. Feitelson; F. Petrini; and J. Fernandez. 2005. "Adaptive Parallel Job Scheduling with Flexible Coscheduling". *IEEE Transactions on Parallel and Distributed Systems*. IEEE Computer Society, Los Alamitos, CA, USA. Vol. 16, No. 11, 1066-1077.

Karatza, H.D. 2001a. "Gang Scheduling Performance under Different Distributions of Gang Size". *Parallel and Distributed Computing Practices*. Nova Science Publishers, Hauppauge, NY, USA, Vol. 4, No. 4, 433-449.

Karatza, H.D. 2001b. "Epoch Scheduling in a Distributed System". In *Proceedings of the Eurosim 2001 Congress* (Delft, Netherlands, Jun.26-19). Eurosim, Delft, Netherlands, 1-6.

Karatza, H.D. 2002. "Scheduling Issues in a Partitionable Parallel System", In *Proceedings of the 16th European Simulation Multiconference* (Darmstadt, Germany, Jun.3-5). SCS Europe, Ghent, Belgium, 522-526.

Li, K. 1997. "An Efficient and Effective Performance Evaluation Method for Multiprogrammed Multiprocessor Systems". In *Proceedings of the 1997 ACM Symposium on Applied Computing* (San Jose, CA, Feb.28–Mar.1). Association for Computing Machinery, New York, N.Y., 478-487.

Setia, S.K. 1997. "Trace-Driven Analysis of Migration-Based Gang Scheduling Policies for Parallel Computers". In *Proceedings of the International Conference on Parallel Processing* (Bloomingdale, IL, Aug.11-15). IEEE Computer Society, Los Alamitos, CA, 489-492.

Strazdins, P. and J. Uhlmann. 2004. "A Comparison of Local and Gang Scheduling on a Beowulf Cluster". In *Proceedings of 2004 IEEE International Conference on Cluster Computing* (San Diego, CA, Sept. 20-23). IEEE Computer Society, Los Alamitos, CA, 55-62.

Streit, A. 2004. "Enhancements to the Decision Process of the Self-Tuning dynP Scheduler". In *Job Scheduling Strategies for Parallel Processing*, *Lecture Notes in Computer Science*, D.G. Feitelson, L. Rudolph and U. Schwiegelshohn (Eds.). Springer-Verlang, Berlin, Germany, Vol. 3277, 63-80.

Wang, F.; M. Papaefthymiou; and M. Squillante. 1997. "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, Vol. 1291, 184-195.

Wiseman, Y. and D. G. Feitelson. 2003. "Paired Gang Scheduling". *IEEE Transactions on Parallel and Distributed Systems*. IEEE Computer Society, Los Alamitos, CA, Vol. 14, No. 6, 581-592.

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

Zhang Y.; A. Yang; A. Sivasubramaniam; and J. Moreira. 2003b. "Gang Scheduling Extensions for I/O Intensive Workloads". In *Job Scheduling Strategies for Parallel Processing, Lecture Notes in Computer Science*, D.G. Feitelson, L. Rudolph and U. Schwiegelshohn (Eds.). Springer-Verlang, Berlin, Heidelberg, Germany. Vol. 2862, 83-207.

AUTHOR BIOGRAPHIES

HELEN D. KARATZA is an Associate Professor in the Department of Informatics at the Aristotle University of Thessaloniki, Greece. Her research interests include Computer Systems Performance Evaluation and Analysis, Multiprocessor Scheduling, Parallel and Distributed Systems, and Simulation. Her web-address is: http://agent.csd.auth.gr/~karatza.