EPOCH TASK SCHEDULING IN DISTRIBUTED SERVER SYSTEMS

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

KEYWORDS
Simulation, Distributed Systems, Scheduling, Performance.

ABSTRACT

In this work we study a special type of task scheduling referred to as epoch scheduling in a distributed server (processor) system. With this policy, processor queues are rearranged only at the end of predefined intervals. The time interval between successive queue rearrangements is called an *epoch*. The objective is to examine if epoch scheduling can perform close to STF method and achieve fairer service than that of STF. A simulation model is used to address performance issues associated with epoch scheduling. Simulated results indicate that epoch scheduling is a good method to use.

INTRODUCTION

Scheduling in distributed server systems has been a major research goal for many years. However, it is not always possible to efficiently execute parallel jobs. To do so, it is necessary to divide programs into tasks, assign the tasks to processors and then schedule them on distributed processors.

Many research papers exist in this research area. For example, few of them are (Abawajy and Dandamudi 2003; Dandamundi 1994; Dandamudi 2003; Harchol-Balter, et al. 2002; Harchol-Balter et al. 2003; Gong and Williamson 2003; Karatza 2000a; Karatza 2000b; Karatza 2002; Karatza and Hilzer 2003; McCann and Zahorjan 1995; Nikolopoulos and Polychronopoulos 2003; Sabin et al. 2003; Weissman, et al. 2003), and many others.

Most research into distributed system scheduling policies has focused on improving system throughput where scheduling overhead is assumed to be negligible. However, scheduling overhead can seriously degrade performance.

FCFS (First Come First Served) is the simplest scheduling method and it is fair to individual jobs but often it results in sub-optimal performance. This method results in no overhead. Many scheduling algorithms have been proposed that achieve higher performance by taking into account information about individual requests.

It is well known that STF (Shortest Task First or Shortest Time First) usually performs best but it has two disadvantages: 1) It involves a considerable amount of overhead because processor queues are rearranged each time new tasks are added. 2) It is possible to starve a task if its service time is large in comparison to the mean service time.

In this work we study epoch scheduling. With this policy, processor queues are rearranged only at the end of predefined intervals. The time interval between successive queue rearrangements is called an epoch. At the end of an epoch, the scheduler recalculates the priorities of all tasks in the system queues using the STF criterion. This type of epoch scheduling is different from epoch scheduling that is studied in (McCann and Zahorjan 1995). In their paper, only policies that provide co-scheduling are considered. Also, in the same paper all nodes are reallocated to jobs at each reallocation point.

In our work we consider that a parallel program has a simple fork-join structure. We do not consider co-scheduling. Rearrangement of queues takes place at the end of predefined intervals, instead of node reallocation to jobs. Epoch scheduling has been also studied in (Karatza 2001; Karatza 2003). The differences between this paper and each of those two papers are the following: In (Karatza 2001) a closed queuing network model is considered with a fixed number of jobs, while in this paper we consider an open queuing network model with various job arrival rates. In (Karatza 2003) the jobs are not parallel and therefore the queues contain jobs, while in this paper each job is parallel and consists of independent tasks which are assigned to queues. Therefore, (Karatza 2003) examines job scheduling, while this paper examines task scheduling.

The results of this study apply to both loosely coupled multiprocessor systems and networks of workstations connected to support parallel applications.

The objective is to study whether we can find an epoch that can perform close to STF but minimizes as much as possible the disadvantages of STF. That is we are interested to find an epoch which combines good performance and that minimizes the number of queue rearrangements and achieves fairer service than that of STF. Performance is examined for different epoch sizes.

Proceedings 18th European Simulation Multiconference
Graham Horton (c) SCS Europe, 2004
ISBN 3-936150-35-4 (book) / ISBN 3-936150-36-2 (CD)

Various workloads are examined. To our knowledge, such an analysis of epoch scheduling has not appeared in the research literature for this type of system operating with our workload models.

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. 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.

For simple systems, performance models can be mathematically analyzed using queuing theory to provide performance measures. However, in the system presented in this paper, fork-join programs and scheduling policies with different complexities are involved. For such complex systems, analytical modelling typically requires additional simplifying assumptions that might have unforeseeable influence on the results. Therefore, research efforts are devoted to finding approximate methods to develop tractable models in special cases, and in conducting simulations.

The precise analysis of fork-join queuing models is a well known intractable problem. For example, (Kumar and Shorey 1993) derived upper and lower bounds for the mean response time when jobs have a linear fork-join structure.

We chose simulations because it is possible to simulate the system in a direct manner, thus lending credibility to the results. Detailed simulation models help determine performance bottlenecks in architecture and assist in refining the system configuration.

The structure of this paper is as follows: Next section specifies system and workload models, it describes scheduling strategies, and it presents the metrics employed while assessing performance of the scheduling policies. Model implementation and input parameters are described in the section after, where also experimental results and performance analysis are presented. The last section contains conclusions and suggestions for further research.

MODEL AND METHODOLOGY

System and Workload Models

This paper uses a simulation model to address scheduling performance issues. An open queuing network model of a distributed server system is considered. P = 16 homogeneous and independent processors are available, each serving its own queue. A high-speed network connects the distributed nodes. The configuration of the model is shown in Figure 1.

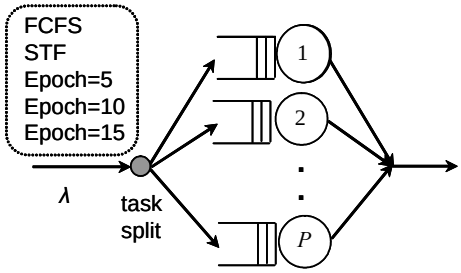

Figure 1: The Queuing Network Model

An important aspect of distributed system design is workload sharing among the processors. This includes partitioning the arriving jobs into tasks that can be executed in parallel, assigning the tasks to processors and scheduling the task execution on each processor. The workload considered here is characterized by three parameters:

- The distribution of job arrival.
- The distribution of the number of tasks per job.
- The distribution of task execution time.

We assume that there is no correlation between the different parameters. For example, a job with a small number of tasks may have a longer execution time.

Job inter-arrival times are exponential random variables with a mean of $1/\lambda$.

Jobs consist of a set of $n \geq 1$ tasks that can be run in parallel. The number of tasks that a job consists of is this job's *degree of parallelism*. It is assumed that the tasks are uniformly distributed in the range of [1..P]. We have chosen the uniform distribution because is one of the distributions that are used for this kind of models. Each task is randomly assigned to a processor queue. Tasks are processed according to the current scheduling method. No migration or pre-emption is permitted.

On completing execution, a task waits at the join point for sibling tasks of the same job to complete execution. Therefore, task synchronization is required, and that synchronization can seriously degrade parallel performance. The price paid for increased parallelism is a synchronization delay that occurs when tasks wait for siblings to finish execution.

The number of tasks of a job j is represented as $t(j)$. Due to the probabilistic assignment of tasks to processor queues, more than one tasks of the same job may be assigned to the same processor. Therefore, if $p(j)$ represents the number of processors required by job j, then the following relation holds:

$$p(j) \leq t(j) \leq P$$

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

Notation used in this paper appears in Table 1.

Job Scheduling Policies

In this work we examine only non-pre-emptive scheduling policies. We assume that the scheduler has perfect information when making decisions, i.e. it knows the execution time of tasks. Next we describe the scheduling strategies employed in this work.

First-Come-First-Served (FCFS). With this strategy, tasks are assigned to a queue in the order of their arrival. This policy is the simplest to implement.

Shortest Task (Time) First (*STF*). This policy assumes that a priori knowledge about a task is available in form of service demand. When such knowledge is available, tasks in the processor queues are ordered in a decreasing order of service demand.

Epoch – x: With this policy, processor queues are rearranged only at the end of predefined time intervals called epochs. The size of an epoch is *x*. At the end of an epoch, the scheduler recalculates the priorities of all tasks in the system queues using the STF criterion.

We study the performance of the Epoch – *x* policy in relation to STF method for different epoch sizes. The goal is to obtain: 1) performance comparable to that of STF, 2) large decrease in the number of queue rearrangements (small overhead), 3) large decrease in the maximum job response time (fairness in individual job service).

It should be noted that a priori information is not often available and only an estimate of task execution time can be obtained. However, it has been reported in the literature (for example in (Dandamudi 1994)) that simulation results have shown that estimation error in processor service times can marginally affect system performance.

Performance Metrics

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

Table 1: Notations

λ	Mean arrival rate of jobs
μ	Mean processor service rate
E	Estimation error in service time
U	Mean processor utilization
RT	Mean Response Time of jobs
MRT	Maximum Response Time of jobs
Synch	Task synchronization time
NQR	Number of Queue Rearrangements
RT Ratio	The ratio of *RT* when SRT or Epoch-*x* method is employed versus *RT* of the FCFS policy
MRT Ratio	The ratio of *MRT* when SRT or Epoch-*x* method is employed versus *MRT* of the FCFS policy
Synch Ratio	The ratio of task synchronization time when SRT or Epoch-*x* method is employed versus *Synch* of the FCFS policy
NQR Ratio	The ratio of *NQR* when the Epoch-*x* method is employed versus *NQR* of the STF policy

RT represents overall performance, while *MRT* expresses fairness in individual job service.

SIMULATION RESULTS AND DISCUSSION

Model Implementation and Input Parameters

The queuing network model described above is implemented with discrete event simulation (Law and Kelton 1991) 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.

We have chosen mean processor service time

$$1/\mu = 1,$$

which means mean service rate per processor $\mu = 1$. Since the processors average 8.5 tasks per job, when all processors are busy, an average of 1.882 jobs are served each unit of time. This implies that the mean job inter-arrival time must be larger than 1/1.882 = 0.531 in order that the system will not be saturated. For this reason we examined the following mean inter-arrival times:

$$1/\lambda = 0.6, 0.625, 0.650, 0.675, 0.7,$$

which means mean arrival rate:

$$\lambda = 1.67, 1.6, 1.54, 1.48, 1.43.$$

Epoch length x was taken as 5, 10, 15. We chose epoch length 5 as a starting point for the experiments because the mean processor service time is equal to 1, and also because with this epoch size the number of queue rearrangements were smaller than in the STF case. Therefore we expected that larger epoch sizes would result in even smaller *NQR*.

Performance Analysis

Figures 2-8 present the performance metrics versus $1/\lambda$. Mean processor utilization U is 0.89, 0.85, 0.82, 0.79, 0.76, for $1/\lambda$ = 0.6, 0.625, 0.650, 0.675, 0.7 respectively (Figure 2).

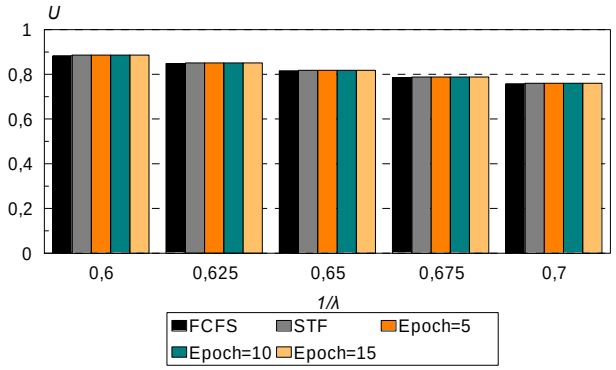

Figure 2. U versus $1/\lambda$

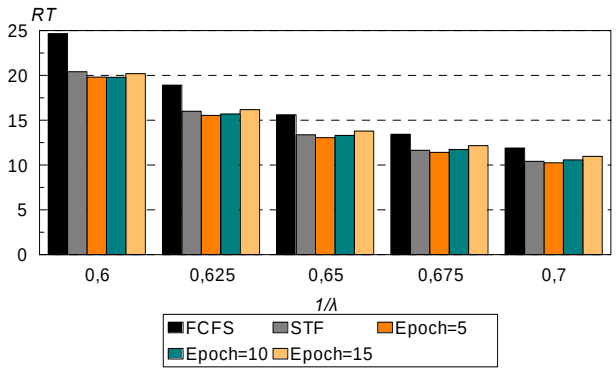

Figure 3. *RT* versus $1/\lambda$

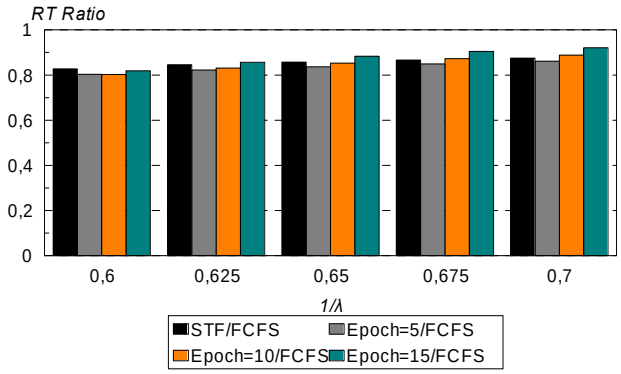

Figure 4. *RT* ratio versus $1/\lambda$

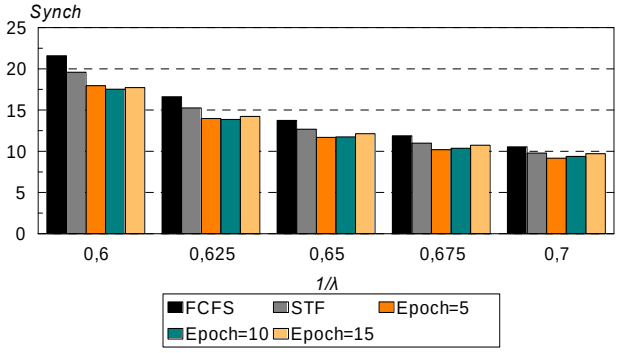

Figure 5. *Synch* versus $1/\lambda$

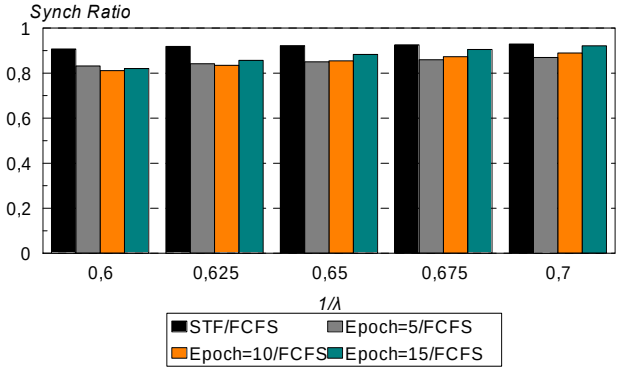

Figure 6. *Synch* ratio versus $1/\lambda$

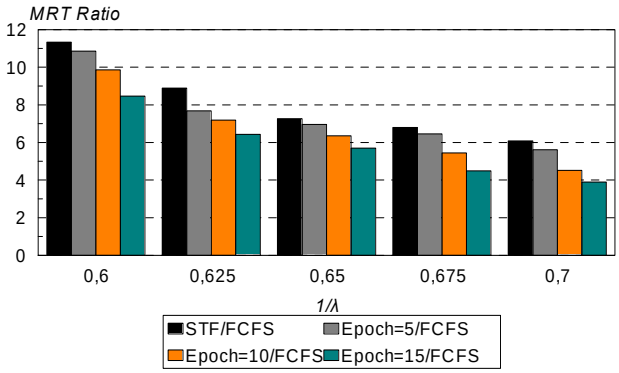

Figure 7. *MRT* ratio versus $1/\lambda$

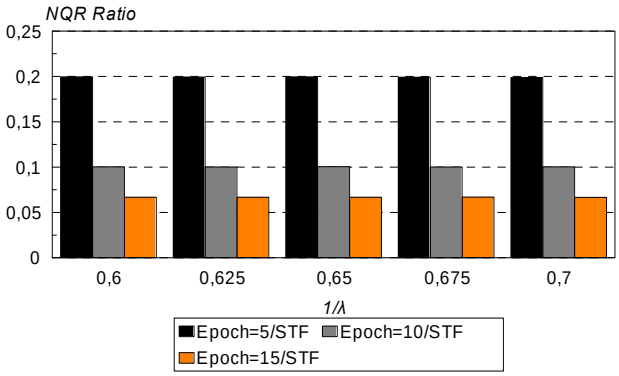

Figure 8. *NQR* ratio versus $1/\lambda$

In Figure 3 it is shown that from all methods examined FCFS method yields the largest response time. This is the reason in all cases *RT* ratio is smaller than one (Figure 4). *RT* ratio increases with increasing mean inter-arrival time which means that the superiority of the STF and Epoch-*x* methods over FCFS is larger at larger loads (that is at smaller mean inter-arrival time). This is because fewer jobs are in the queues when $1/\lambda$ is large than when is small, so there are then fewer opportunities to exploit the advantages of the STF and Epoch-*x* methods.

RT generally increases with increasing epoch size, but the increase is not significant. For this reason *RT* ratio slightly increases with increasing epoch size.

In all cases examined, epoch scheduling for epoch size 5 yields slightly smaller *RT* than STF. Epoch-10 yields *RT* which is very close to the *RT* of the STF case. Epoch=15 performs either close (at larger loads) or a little worst (at smaller) loads than the STF method (Figures 3, and 4). However, even with this epoch size the difference in performance between epoch scheduling and STF is not significant.

Figure 5 shows that with all scheduling methods task synchronization time is decreasing with decreasing load. This is because it is more probable at large loads than at small loads for the first task of a job that finishes execution to wait for a long time some sibling tasks that are still waiting in processor queues.

It is also shown in Figure 5 that the largest *Synch* incurs with the FCFS method. This is because it is more probable when the FCFS policy is employed than when one of the other scheduling strategies are used, some small tasks to wait behind some large tasks in processor queues. A consequence of this may be long syncronization delay of sibling tasks. On the other hand, the STF method causes delays to large tasks, but the simulation results reveal that these delays influence *Synch* in a smaller degree than the delays caused by the FCFS policy. *Synch* is larger in the STF case than in the epoch scheduling case, as in the later case priority is given to small tasks on a periodic basis only.

In Figure 6 it is shown that the difference in *Synch* between FCFS and each of the remaining methods is generally larger at larger loads. This is because regarding synchronization delay of sibling tasks, the advantages of STF and epoch scheduling over FCFS are better exploited when there are many job tasks in the system than when there are few.

In Figure 7 it is shown that in all cases the STF method yields the largest *MRT*, while the smallest *MRT* is produced by the FCFS method. In all cases, epoch scheduling yields smaller maximum response time as compared to the STF method. Therefore, in all cases epoch scheduling is fairer than STF. Furthermore, Figure 7 shows that in all cases epoch scheduling is fairer when epoch size is large than when is small.

MRT ratio is decreasing with increasing mean inter-arrival time. This is due to the fact that when STF or epoch scheduling is employed, more job tasks go out of order for longer time when the load is large than when is small. Therefore, larger *MRT* can be produced at large loads than at small ones.

Figure 8 presents the *NQR* ratio. It is shown that for all λ the decrease in the number of queue rearrangements due to epoch scheduling is very high. Therefore significant reduction in the number of queue rearrangements can be achieved when epoch scheduling is employed instead of STF. In the same Figure is also shown that for each epoch size *NQR* ratio is nearly the same for all values of λ. The *NQR* ratio is about 0.2, 0.10, and 0.07 in the Epoch=5, 10, and 15 cases respectively. For each λ, *NQR* ratio decreases with increasing epoch size which means that *NQR* is smaller when epoch size is large than when is small.

In order to study the impact of service time estimation error on the performance of the STF and Epoch-*x* methods, additional simulation experiments where conducted. In those experiments task execution time estimated was assumed to be uniformly distributed within ± E% of the exact value. We set estimation error at ±10%. The results showed that the estimation error did not significantly affect performance. This is in accordance with other results in the literature related to estimation of service time (Dandamudi 1994). For this reason in this paper we present the results for the exact service times only. This means that we consider estimation error set at ±0%.

CONCLUSIONS AND RECOMMENDED FUTURE RESEARCH

This paper studies task scheduling policies in a distributed server system. Simulation is used to generate results used to compare different configurations.

Simulation results reveal that epoch task scheduling is a good policy to choose. For the epoch lengths that we examined this policy performs very close to STF. Furthermore and more importantly, it involves much less overhead and is also fairer than STF.

It is also shown that all of the epoch lengths have merit. For all loads that we examined large epochs result in fairer service for individual jobs, and involve less overhead than short epochs. On the other hand, response time is shorter with short epochs. However, for the epoch lengths that we examined, response time does not differ significantly at different epochs.

This paper represents a case study where the number of tasks per job is bounded by the number of distributed

servers in the system (the uniform distribution is used). As a future research we plan to consider the exponential distribution for the number of tasks per job, so that we can study the performance of epoch scheduling when the number of job tasks can be larger than the number of processors.

REFERENCES

Abawajy, J.H. and S. Dandamudi. 2003. "Scheduling Parallel Jobs with CPU and I/O Resource Requirements in Cluster Computing Systems". In *Proceedings of the 11th IEEE/ACM International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunications Systems (Mascots'03)* (Orlando, FL, Oct.12-14). IEEE Computer Society, Los Alamitos, CA, 336-343.

Dandamudi, S. 1994. "Performance Implications of Task Routing and Task Scheduling Strategies for Multiprocessor Systems". In *Proceedings of the IEEE-Euromicro Conference on Massively Parallel Computing Systems* (Ischia, Italy, May 2-6). IEEE Computer Society, Los Alamitos, CA, 348-353.

Dandamudi, S. 2003. *Hierarchical Scheduling in Parallel and Cluster Systems*. 1rst edn. Kluwer Academic/Plenum Publishers, New York.

Harchol-Balter, M.; K. Sigman; and A. Wierman. 2002. "Asymptotic Convergence of Scheduling Policies with Respect to Slowdown". In *Proceedings of IFIP Performance 2002* (Rome, Italy, Sept.22-27). *Performance Evaluation* 49, Elsevier B.V., Amsterdam, The Netherlands, 241-256.

Harchol-Balter, M.; B. Schroeder; N. Bansal; and M. Agrawal. 2003. "Size-based Scheduling to Improve Web Performance". *ACM Transactions on Computer Systems* 21, No.2, Association for Computing Machinery, New York, N.Y., 1-27.

Gong, M. and C. Williamson. 2003. "Quantifying the Properties of SRPT Scheduling". In *Proceedings of the 11th IEEE/ACM International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunications Systems (Mascots'03)* (Orlando, FL, Oct.12-14). IEEE Computer Society, Los Alamitos, CA, 126-135.

Karatza, H.D. 2000a. "Scheduling Strategies for Multitasking in a Distributed System". in *Proceedings of the 33rd Annual Simulation Symposium* (Washington, D.C., Apr.16-20). IEEE Computer Society, Los Alamitos, CA, pp. 83-90.

Karatza, H.D. 2000b. "A Comparative Analysis of Scheduling Policies in a Distributed System using Simulation". *International Journal of Simulation: Systems, Science & Technology* 1, No.1-2 (Dec.), UK Simulation Society, 12-20.

Karatza, H.D. 2001. "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. "Task Scheduling Performance in Distributed Systems with Time Varying Workload". *Neural, Parallel & Scientific Computations* 10, No. 3, Dynamic Publishers, Atlanta, GA, 325-338.

Karatza, H.D. 2003. "A Comparison of Load Sharing and Job Scheduling in a Network of Workstations". *International Journal of Simulation: Systems, Science Technology* 4, No. 3&4, UK Simulation Society, Nottingham, UK, 4-11.

Karatza, H.D. and R.C. Hilzer. 2003. "Parallel Job Scheduling in Distributed Systems". *Simulation: Transactions of the Society for Modeling and Simulation International* 79, No.5 (May), Sage Publications, Thousand Oaks, CA, 287-298.

Kumar A. and R. Shorey. 1993. "Performance Analysis and Scheduling of Stochastic Fork-Join Jobs in a Multicomputer System". *IEEE Transactions on Parallel and Distributed Systems* 4, No.10, IEEE Piscataway, NJ, 1147-1162.

McCann, C. and J. Zahorjan. 1995. "Scheduling Memory Constrained Jobs on Distributed Memory Parallel Computers". In *Proceedings of the 1995 ACM Sigmetrics Conference* (Ottawa, Canada, May 15-19). The Association for Computing Machinery, New York, 208-219.

Law A. and D. Kelton. 1991. *Simulation Modelling and Analysis*. 2nd Ed., McGraw-Hill, New York, USA.

Nikolopoulos, D.S. and C.D. Polychronopoulos. 2003. "Adaptive Scheduling Under Memory Constraints on Non-Dedicated Computational Farms". *Future Generation Computer Systems* 19, Elsevier, Amsterdam, 505–519.

Sabin, G.; R. Kettimuthu; A. Rajan; and P. Sadayappan. 2003. "Scheduling of Parallel Jobs in a Heterogeneous Multi-Site Environment". In *Job Scheduling Strategies for Parallel Processing, Lecture Notes in Computer Science* 2862, D. Feitelson, L. Rudolph, and W. Schwiegelshohn (Eds.). Springer-Verlag, Berlin Heidelberg, 87-104.

Weissman, J.B.; L.R. Abburi; and D. England. 2003. "Integrated Scheduling: the Best of Both Worlds". *Journal of Parallel and Distributed Computing* 63, Elsevier Science, New York, USA, 649–668.

AUTHOR BIOGRAPHY

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, Parallel and Distributed Systems Scheduling, and Simulation. Dr. Karatza is an Associate Editor of SIMULATION: Transactions of the Society for Modeling and Simulation International, and a member of the Editorial Board of the International Journal of Simulation: Systems, Science & Technology, and Simulation Modelling Practice and Theory Journal. Her email address is: karatza@csd.uth.gr and her Web-page can be found at http://www.csd.auth.gr/~karatza.