A COMPARISON OF LOAD SHARING AND JOB SCHEDULING IN A NETWORK OF WORKSTATIONS

HELEN D. KARATZA

Department of Informatics
Aristotle University of Thessaloniki
54006 Thessaloniki, GREECE
Email: karatza@csd.auth.gr

Abstract: This paper examines load sharing and job scheduling in a network of workstations (NOW). Along with traditional methods of load sharing and job scheduling, it also examines methods referred to as epoch load sharing and epoch scheduling respectively. Epoch load sharing evenly distributes the load among workstations with job migration that occurs only at the end of predefined intervals. The time interval between successive load sharing events is called an epoch. The objective is to reduce the number of times that global system information is needed to make allocation decisions, while at the same time achieving good overall performance and fairness of individual job service. In the epoch scheduling case, processor queues are rearranged only at the end of epochs. The objective is to find an epoch that minimizes the number of queue rearrangements, while being fair and producing good system performance. Simulation is used to measure performance issues associated with load sharing and job scheduling.

Keywords: Simulation, Networks of Workstations, Load Sharing, and Scheduling.

1. INTRODUCTION

Networks of workstations provide attractive scalability in terms of computation power and memory size. As they become viable platforms for a wide range of applications, new policies are needed to allocate workstation resources to competing users and also to schedule jobs in the queues.

Generally, no processor in a distributed system should remain idle while others are overloaded. It is preferable that the workload be uniformly distributed over all processors. It is important to efficiently utilize computational power.

The purpose of load sharing is to ensure that no processor remains idle, particularly when there other processors are heavily loaded. With sender-initiated algorithms, load-distribution activity is initiated when an over-loaded node (sender) tries to send a task to another under-loaded node (receiver). In receiver-initiated algorithms, an under-loaded node (receiver) initiates load-distribution when a task is requested from an over-loaded node (sender).

Load sharing policies that use information about the average behavior of the system and ignore the current state, are called static policies. Static policies may be either deterministic or probabilistic. Policies that react to the system state are called adaptive or dynamic policies. Dynamic load sharing is an important system function designed to distribute workload among available processors and improve overall performance.

The principle advantage of static policies is simplicity, since they do not require the maintenance and processing of system state information.

Adaptive policies are more complex, mainly because they require information on the system's current state when making transfer decisions. However, the added complexity often produces significantly better performance than static policies. This paper investigates probabilistic, deterministic and adaptive load sharing policies.

In the probabilistic case, scheduling policies are described by state independent branching probabilities. Jobs are dispatched randomly to workstations with equal probability. In the deterministic case, jobs join the shortest workstation queue since routing decisions are based on system state.

The adaptive case uses a variation of the probabilistic policy. When workstations become idle, jobs can migrate from heavily loaded workstation queues to idle workstations. This is a receiver initiated adaptive load sharing method. It balances the job load and can improve overall system performance.

Epoch load sharing is another adaptive load sharing method. It evenly distributes the load among workstations and uses job migration only at the end of prede-

fined intervals. The time interval between successive load sharing transfers is called an epoch.

Most distributed system load sharing research focuses on improving some performance metrics such as mean response time where scheduling overhead is assumed to be negligible. However, scheduling overhead can seriously degrade performance. Therefore, the number of times the scheduler is called to make load sharing decisions can be a factor that degrades performance.

Also, the transfer of jobs to remote workstations incur communication costs. In this model, only queued jobs are transferred. It is assumed that the job being executed is not eligible for transfer because doing so is complex. An obvious disadvantage of pre-emptive migration is the need to transfer the memory associated with the migrated process; thus, migration costs for an active process is much greater than the cost of remote execution.

We compare epoch load sharing with the other load sharing methods. The objective is to reduce the number of times that global system information is needed to make allocation decisions, while at same time achieving good performance.

It is well known that shortest queue routing performs better than the probabilistic. However, the shortest queue method invokes the scheduler each time a job demands processing service. We aim to find an epoch that performs well in comparison with the shortest queue method and involves minimal overhead. In this work, migration overhead impacts on performance. Therefore, scheduling optimality minimizes the number of times the scheduler has been activated, since invoking the scheduler requires considerable overhead to collect load information from all of the workstations.

Load sharing and load balancing are topics that have already been studied by many authors ([Blake, 1992], [Eager et al, 1986], [Harchol-Balter and Downey, 1996], [Karatza, 1996a], [Karatza, 1996b], and [Karatza, 1998]). However, these studies do not address epoch job migration.

The epoch load sharing examined here is different from the epoch scheduling studied in [McCann and Zahorjan, 1995]. That paper, focuses on scheduling memory-constrained jobs in distributed memory parallel systems. Only co-scheduling policies are considered, and the number of processors allocated to a job may change during execution. All nodes are reallocated jobs at each reallocation point.

We do not consider co-scheduling. Instead of reallocating nodes, we consider load sharing via job migration at the end of predefined intervals.

Periodic load balancing is studied in [Hjalmtysson and Whitt, 1998a], and [Hjalmtysson and Whitt, 1998b]. The main contributions of these papers are analytical models and formulas describing the performance of periodic load balancing. Their goal is to describe the distribution of the workload at each queue as a function of time, especially just before and just after each reconfiguration (balancing).

When a job is assigned a workstation queue, it is scheduled for execution accordingly to the scheduling method employed. Scheduling improvement has been a major research and development goal for several years ([Dandamudi, 1994], and [Karatza, 2000]). In [Dandamudi, 1994] it is shown that scheduling policies have substantial impact on performance when non-adaptive routing strategies are used. This is why we examine job scheduling in the workstation queues only when the probabilistic routing is applied.

The First-Come-First-Served (FCFS) method is the simplest scheduling strategy. It is fair to individual jobs but often yields sub-optimal performance. This method incurs no overhead. Many proposed scheduling algorithms achieve higher performance by considering information about individual requests.

The Shortest-Job-First (SJF) policy usually performs best but it has the following two disadvantages: a) It involves a considerable amount of overhead because processor queues are rearranged each time new jobs are added. b) It is possible to starve a job if its service time is large in comparison to the mean service time.

We consider the FCFS and SJF scheduling methods, as well as epoch scheduling where the scheduler recalculates priorities of all jobs in the system queues using the SJF criterion at the end of each epoch.

The goal is to find an epoch that is close to optimal but minimizes the disadvantages of SJF. Scheduling optimality is defined to be a minimization of queue rearrangements, which is equivalent to maximizing the size of the epochs.

We compare cases of load sharing and job scheduling for various workloads and different epoch sizes. Fairness among competing jobs is required. Comparative results are generated using simulation techniques

Epoch load sharing was previously studied in [Karatza and Hilzer, 2001], and [Karatza and Hilzer, 2002]. Epoch scheduling was studied in [Karatza, 2001]. However, the system and workload models are differ-

ent from the models examined here. The other papers consider closed queuing network models with fixed numbers of jobs. Also none of them compares load sharing and job scheduling strategies. They consider these two issues separately, in different papers.

An open queuing network models the network of workstations in this paper. Simulation indicates that epoch load sharing often produces good overall performance and is fair in job service. To our knowledge, no other comparative analysis of load sharing and job scheduling that include epoch load sharing and epoch scheduling appears in research literature.

This paper is structured as follows. Section 2.1 specifies system and workload models, sections 2.2 and 2.3 describe the load sharing and scheduling strategies, respectively, and section 2.4 presents the metrics employed while assessing the performance of these strategies. 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 conclusions and suggestions for further research.

2. MODEL AND METHODOLOGY

2.1 System and Workload Models

An open queuing network model for a NOW is considered. $P = 16$ homogeneous workstations are available, each serving its own queue.

A high-speed network connects the distributed nodes. This is a representative model for many existing departmental networks of workstations.

The configuration of the model is shown in Figure 1.

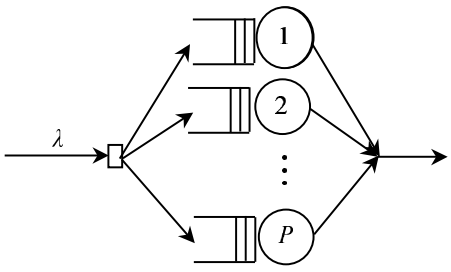

Figure 1. The queuing network model

The jobs examined are highly independent. For example, once a job commences execution, no job ever idly waits for communication with (i.e., synchronizes with) other jobs.

We consider the problem of load sharing and also the problem of job scheduling.

In the case of load sharing, when a job is transferred to a workstation for remote processing, the job incurs communication costs. Only jobs waiting in the queues are transferred.

A latest-job-arrived policy is used to select a job for transfer from the sending to the receiving workstation. We believe that the average transfer cost for non-executing jobs, although no negligible, is quite low relative to average job processing costs. The benefits of migration depend on migration costs.

The workload considered here is characterized by three parameters:

- The distribution of job service demand.
- The distribution of job arrival.
- The distribution of the communication overhead.

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

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

The communication channel is modeled as a single server queuing system, whose service time is an exponentially distributed random variable with mean *Co,* in order to deal with the effects of communication overhead.

Notation used in this paper appears in Table 1.

2.2 Load Sharing Policies

The following load sharing strategies are employed in our simulations. The scheduling policy in each queue is FCFS.

***Probabilistic* (*Pr*)**

With this policy, a job is dispatched randomly to one of the workstations with equal probability. Therefore, with this method, the scheduler is never activated to make decisions that depend on system state.

***Probabilistic with Migration* (*PrM*)**

Jobs are assigned to processor queues in the same way as the *Pr* case. However, when a processor becomes idle and there are jobs waiting at the other processor queues, a job can migrate from the most heavily loaded processor to the idle processor. Since load-distribution is initiated by an idle node (receiver) which tries to get a job from an overloaded node

(sender), this is a receiver-initiated algorithm. For stability reasons, sender nodes must have a queue length greater than one. The scheduler is activated only when a processor becomes idle.

Shortest Queue (SQ)

This strategy assigns a job to the shortest processor queue. Therefore, the scheduler is activated every time a job arrives.

Epoch Load Sharing (ELS)

This policy uses migration at the end of epochs to distribute the load evenly among workstations. At the epochs, the scheduler collects information about the status of all workstation queues, evaluates the mean of all queue lengths, and places processor queue lengths into increasing order in a table. Then, it moves jobs from the most heavily loaded to lightly loaded processors until either all processors have queue lengths equal to the mean or differ at most by one job.

All four of the above load-sharing schemes have merit.

Pr is the simplest method since it involves only a negligible amount of overhead to generate random numbers, but Pr produces sub-optimal performance. It never activates the scheduler, as it does not make decisions that depend on system state. The migratory version of this policy invokes the scheduler when a processor becomes idle and it involves overhead each time a job migrates.

The SQ method requires global knowledge of queues on job arrival, so the scheduler is called upon every time a job arrives.

Migration overhead is taken into account in this study. Therefore, a major concern is the number of times the system scheduler is called to collect information about processor queues in order to manage the information and to make transfer decisions, which is the case of epoch load sharing.

2.3 Job Scheduling Policies

Next the scheduling policies used by probabilistic routing are described.

First-Come-First-Served (FCFS)

This strategy schedules jobs in the order of their arrival. It is the simplest policy to implement and also is the fairest scheduling method. Since probabilistic routing is applied, this strategy is the same as Pr.

Shortest-Job-First (SJF)

SJF requires a priori knowledge of job service demand. Jobs are ordered in a decreasing order of service demand in processor queues. However, advance service time information is not always available and so only an estimate of job execution time be used instead. For this reason, it is assumed that estimated service time is uniformly distributed within $\pm E\%$ of the exact value.

Epoch Scheduling (ES)

Processor queues are only rearranged at the end of epochs with this policy. At each epoch, the scheduler recalculates the priorities of all jobs in system queues using the SJF criterion.

Table 1. Notations

Co	Mean communication delay due to job migration
λ	Mean arrival rate
μ	Mean processor service rate
RT	Mean Response Time
MW	Maximum Wait time
NSA	Number of system Scheduler Activations to collect information about processor queues
NQR	Number of Queue Rearrangements
E	Estimation error in service time
D_{RT}	Relative (%) decrease in RT when one of the above described methods is employed instead of the Pr (FCFS) policy
D_{NSA}	Relative (%) decrease in NSA when one of the above described load sharing methods is employed instead of the SQ policy
D_{NQR}	Relative (%) decrease in NQR when ES is employed instead of the SJF policy
MW Ratio	The ratio of MW in a case X over MW in another case Y

It is obvious that processor queues must be rearranged each time new jobs are added in the SJF case. Also, SJF is unfair to jobs with large service demands. The goal of ES is to decrease the number of queue rearrangements in comparison with the SJF method, to provide fairness and to yield good performance.

2.4 Performance Metrics

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

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

When load sharing is examined, case X is Pr, while case Y is one of the remaining load sharing methods. When job scheduling is examined, case X is FCFS, while case Y is one of the other job scheduling methods. Load sharing and job scheduling use the same measure for comparisons in this paper because FCFS is actually the Pr method.

3. SIMULATION RESULTS AND DISCUSSION

3.1 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 there are 16 workstations, we chose $\lambda < 16$, and consequently $1/\lambda > 0.0625$. For this reason we examined the following mean inter-arrival times:

$$1/\lambda = 0.07, 0.08, 0.09,$$

which corresponds to the following arrival rates:

$$\lambda = 14.286, 12.5, 11.11.$$

Epoch sizes are 2, 4, 6, and 8. We chose epoch length 2 as a starting point because the mean processor service time is equal to 1, and because with this epoch size *NSA* is much smaller than in the SQ case. We expect that larger epoch sizes would result in even smaller *NSA*. Also, with this epoch size, *NQR* is smaller than in the SJF case. Therefore, it is expected that larger epoch sizes would result in even smaller *NQR*.

$Co = 0.1$ is considered for mean migration cost. In all cases, service time estimation errors of ±10% and ±30% are considered.

3.2 Performance Analysis

The following results represent performance of the different policies.

In each of these cases, system load (mean workstation utilization) is:

0.88 for $1/\lambda = 0.07$,

0.77 for $1/\lambda = 0.08$,

0.69 for $1/\lambda = 0.09$.

3.2.1 Load Sharing Performance

Figures 2, 3, 4, 5 present the performance metrics versus $1/\lambda$ (mean inter-arrival time) in the load sharing case.

In all cases, the overall performance in terms of mean response time is superior with the SQ method. The superiority of SQ is intuitive. However, the intent of this study is to determine how superior it is to other methods, and whether the extent of its superiority justifies overhead required to maintaining knowledge of processor queue lengths.

Pr is worst method. The difference in *RT* between Pr and each of the other methods decreases with decreasing arrival rate λ.

The migration of jobs when using PrM significantly improves overall performance as compared with Pr. This is because when Pr is used, there are cases where processors have unbalanced queues and therefore may benefit from job migration. For all λ, the difference in performance between PrM and Pr varies between 51% and 75%.

For all epoch sizes, epoch load sharing performs significantly better than Pr. Mean response time performance improves with decreasing epoch size. However, the PrM algorithm performs better than ELS. For all λ, the difference between ELS and each of PrM, and SQ increases with increasing epoch size. Generally, the superiority of SQ over PrM and ELS increases with increasing mean inter-arrival time.

For all λ, the relative decrease in the number of scheduler activations (D_{NSA}) is very high for all epoch sizes (i.e., in the range of 96-99%). For any λ, D_{NSA} increases slightly with increasing epoch size. D_{NSA} is lower in the PrM case (in the range 70% – 76%). The difference in the number of scheduler activations between PrM and ELS slightly decreases with decreasing λ.

Figure 5 shows that the *MW* Ratio decreases with decreasing λ with each method. *MW* Ratio > 1.0 holds in

all cases. Therefore, for all load-sharing methods, *Pr* yields the highest maximum wait time. Lower *MW* is produced by the SQ policy. The *MW* Ratio in the PrM case is lower than in the SQ case but it is higher than the ELS case. *MW* Ratio decreases with increasing epoch size. For all λ, ELS for epoch size 2 yields a *MW* that is not significantly larger than that in the PrM case.

3.2.2 Job Scheduling Performance

Figures 6, 7, 8, 9 present the performance metrics versus $1/\lambda$ in the job scheduling case.

As was expected, the lowest mean response time is produced by SJF (Figure 6) and Epoch scheduling yields lower mean response time than FCFS. For any λ, *RT* increases with increasing epoch size.

With all methods, D_{RT} decreases with increasing mean inter-arrival time. This is because fewer jobs are in the queues when $1/\lambda$ is large, so there are fewer opportunities to exploit the advantages of the SJF and ES methods. For any λ, D_{RT} decreases with increasing epoch size.

For all λ, the relative decrease in the number of queue rearrangements due to epoch scheduling is very high in comparison to the decrease in mean response time. For example, D_{NQR} varies in the range of 51- 87% while D_{RT} varies in the range of 8 - 53%.

D_{NQR} increases with increasing epoch size. For each epoch size, D_{NQR} is nearly the same for all λ.

SJF and ES yield larger maximum wait times than FCFS. ES for epoch length 2 yields the same *MW* Ratio as the SJF policy. For any λ, *MW* Ratio increases with increasing epoch size. However, as λ decreases, the *MW* Ratio for all epoch sizes tends to approach *MW* for the SJF case.

Additional simulation experiments access the impact of service time estimation error on the performance of the scheduling methods.

The simulation results indicate that estimation error in processor service times marginally affect system performance. Therefore, there is no gain from a priori knowledge of exact service times. This is the why the results presented here refer only to the $E = 0$ case.

3.2.3 Load Sharing Performance versus Job Scheduling Performance

It is apparent that PrM, SQ and ELS for epoch sizes 2, 4, and 6 yield smaller *RT* than SJF does, and SJF is only slightly better than ELS for epoch size 8.

With regard to fairness of individual job service, the maximum wait time in the SJF and ES cases is much higher than in the load sharing cases.

However, when load sharing policies and job scheduling methods are compared, we notice that they involve different overhead.

The load sharing policies require global system state information. For large distributed systems collecting and processing global system state information is not trivial. On the other hand, job scheduling methods rearrange the sequencing of jobs in each queue without any knowledge of other queue status. However, job scheduling requires advance knowledge of approximate job service demand, which is not always available.

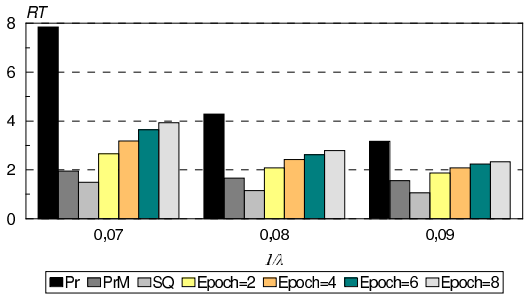

Figure 2. *RT* versus $1/\lambda$, Load Sharing

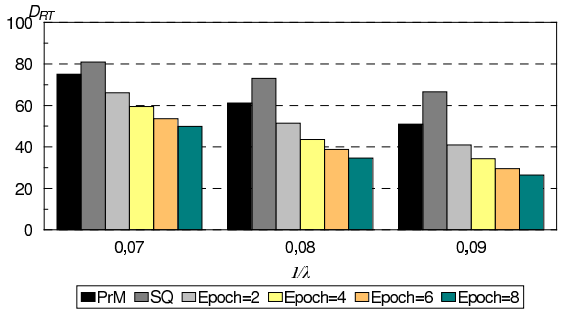

Figure 3. D_{RT} versus $1/\lambda$, Load Sharing

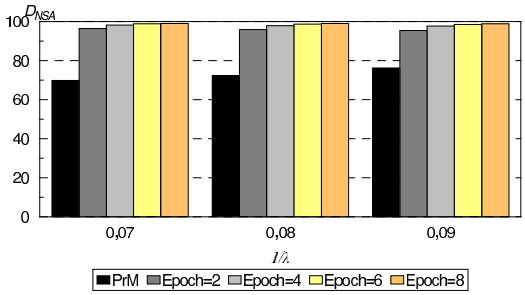

Figure 4. D_{NSA} versus $1/\lambda$, Load Sharing

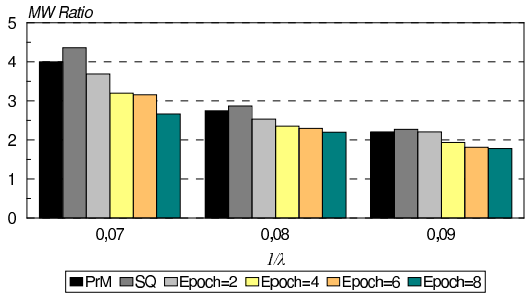

Figure 5. *MW Ratio* versus 1/λ, Load Sharing

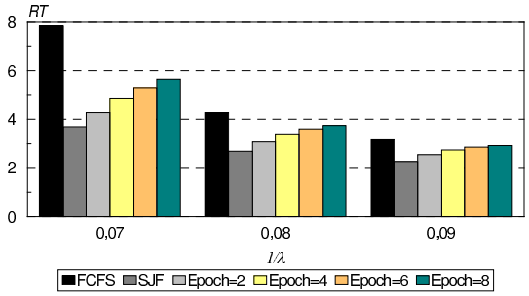

Figure 6. *RT* versus 1/λ, Job Scheduling

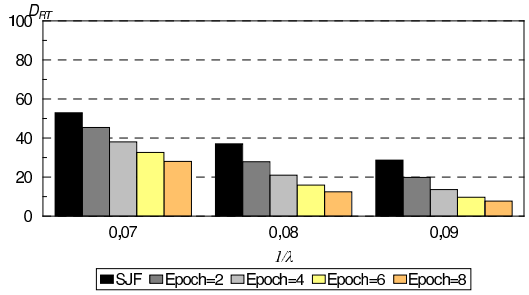

Figure 7. D_{RT} versus 1/λ, Job Scheduling

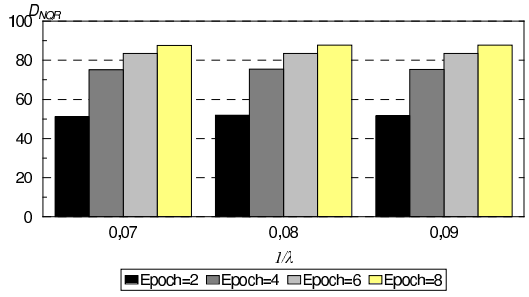

Figure 8. D_{NQR} versus 1/λ, Job Scheduling

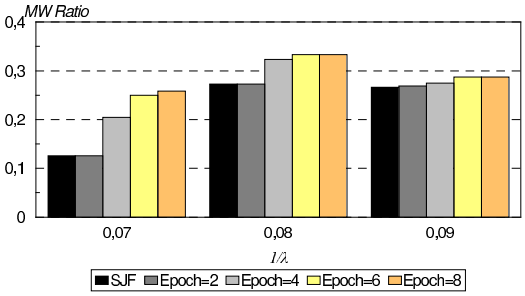

Figure 9. *MW Ratio* versus 1/λ, Job Scheduling

4. CONCLUSIONS AND RECOMMENDED FUTURE RESEARCH

This paper studies load sharing and job scheduling policies in a network of workstations. Simulation is used to generate results used to compare different configurations.

Epoch Load Sharing (ELS) and Epoch Scheduling (ES) are studied along with the Probabilistic (Pr), Probabilistic with Migration (PrM), and Shortest Queue (SQ) load sharing policies, and the FCFS and Shortest-Job-First (SJF) job scheduling methods. Simulation results reveal the following:

PrM, SQ and ELS for small epoch sizes perform better than SJF and ES in terms of overall performance and fairness.

SQ is the best method only if overhead is not taken into account. ELS with small epoch size is preferred as it performs close to SQ and produces much less overhead than PrM and SQ.

However, when global system state information is not available, ES with a small epoch size should be used as it performs closer to SJF than when the epoch size is large, and it involves less overhead than SJF.

Future work will compare load sharing and job scheduling policies in heterogeneous NOWs.

REFERENCES

Blake B.A. 1992, "Assignment of Independent Tasks to Minimize Completion Time". *Software-Practice and Experience*, John Wiley & Sons, Inc., New York, USA, Vol.22 (9). Pp723-734.

Dandamudi S. 1994, "Performance Implications of Task Routing and Task Scheduling Strategies for Multiprocessor Systems". In *Proc. of the IEEE-Euromicro Conference on Massively Parallel Com-*

puting Systems (Ischia, Italy, May) IEEE Computer Society, Los Alamitos, CA, USA. Pp348-353.

Eager D.L., Lazowska E.D. and Zahorjan J. 1986, "A Comparison of Receiver-Initiated and Sender-Initiated Adaptive Load Sharing". *Performance Evaluation*, Elsevier, Amsterdam, Holland, Vol. 6. Pp53-68.

Harchol-Balter M. and Downey A.B. 1996, "Exploiting Process Lifetime Distribution for Dynamic Load Balancing". In *Proc. of Sigmetrics '96* (Philadelphia, PA, USA, May) ACM, New York, USA. Pp13-24.

Hjalmtysson G. and Whitt W. 1998a, "Approximations for Periodic Load Balancing". In *Proc. of the 37th IEEE Conference on Decision and Control* (Tampa, Florida, USA, December) IEEE Computer Society, Los Alamitos, CA, USA. Pp35-41.

Hjalmtysson G. and Whitt W. 1998b, "Periodic Load Balancing". *Queueing Systems*, Kluwer Academic Publishers, New York, USA. Vol. 30 (1-2), Pp203-250.

Karatza H.D. 1996a, "Simulation Study of Sender-Initiated Load Sharing with Resequencing". In *Proc. of Summer Computer Simulation Conference* (Portland, Oregon, USA, July) SCS, San Diego, CA, USA. Pp497-501.

Karatza H.D. 1996b, "Sender-Initiated versus Receiver-Initiated Adaptive Load Sharing with Resequencing". In *Proc. of the 8th European Simulation Symposium & Exhibition* (Genoa, Italy, October) SCS Europe, Ghent, Belgium. Pp 546-550.

Karatza H.D. 1998, "Assignment of Programs in a Distributed System with Resequencing". In *Proc. of the 31th Annual Simulation Symposium* (Boston, MA, USA, April) IEEE Computer Society, Los Alamitos, CA, USA. Pp34-41.

Karatza H.D. 2000, "A Comparative Analysis of Scheduling Policies in a Distributed System using Simulation". *International Journal of Simulation: Systems, Science & Technology*, UK Simulation Society, Nottingham, England, Vol. 1 (1-2). Pp12-20.

Karatza H.D. and Hilzer R.H. 2001, "Epoch Load Sharing in a Network of Workstations". In *Proc. of the 34th Annual Simulation Symposium* (Seattle, WA, USA, April) IEEE Computer Society, Los Alamitos, CA, USA. Pp36-42.

Karatza H.D. 2001, "Epoch Scheduling in a Distributed System". In *Proc. of the Eurosim 2001 Congress* (Delft, Netherlands, June) Eurosim, Vienna, Austria. Pp1-6.

Karatza H.D. and Hilzer R.H. 2002, "A Simulation-Based Performance Analysis of Epoch Load Sharing in Distributed Systems". *Transactions of the Society for Modeling and Simulation International*, SCS, San Diego, CA, USA, Vol. 78 (7). Pp461-471.

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

McCann C. and Zahorjan J. 1995, "Scheduling Memory Constraint Jobs on Distributed Memory Parallel Computers". In *Proc. of the 1995 ACM Sigmetrics Conference* (Ottawa, Canada, May) ACM, New York, USA. Pp208-219.

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.