Assignment of Programs in a Distributed System with Resequencing

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

Abstract

The assignment of independent applications in a distributed system in conjunction with resequencing is studied in this paper using simulation techniques. Three scheduling policies are studied and compared. A probabilistic policy, and two policies that use an estimation of job service time, one deterministic and one adaptive. Their performance is examined for a variety of workloads. The impact on performance of resequencing delay is also explored.

1. Introduction

Distributed computing systems are composed of several loosely-coupled processors communicating over a network. The area of distributed systems has been the focus of a tremendous amount of research in the last decade. Distributed computer systems have many advantages, including the capability to share processing of tasks in the event of overloads and failure, transparency, and modularity.

In a distributed system a job might wait for service at the queue of one processor while at the same time another processor capable of serving the job is idle. It is therefore desirable to achieve a uniform distribution of the workload among the processors in such a way that, there will be no idle processors while some processors are overloaded.

In a distributed system, it is required that relevant information for the assignment decision, such as processor loads, must be automatically available to the operating system on all processors.

Load distribution is necessary in a distributed system for better utilizing its computational power. Various load balancing and load sharing algorithms appear in the literature. In general, the purpose of the load balancing operation is to split the work evenly among the processors, whereas the approach of load sharing algorithms is to ensure that no processor stays idle loaded when there are heavily loaded processors in the system. Under sender-initiated algorithms, load-distributing activity is initiated by an overloaded node (sender) trying to send a task to an underloaded node (receiver). In receiver-initiated algorithms, load distributing activity is initiated from an underloaded node (receiver), which tries to get a task from an overloaded node (sender).

Scheduling policies that only use information about the average behaviour of the system, ignoring 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 policies.

The principle advantage of static policies is their simplicity. There is no need to maintain and process system state information. Adaptive policies, by contrast, are more complex, since they employ information on the current system state in making transfer decisions. This information makes possible significantly greater performance benefits that can be achieved under static policies.

In this paper we study the effects of job scheduling and resequencing on system and program performance of a distributed system which consists of two identical processors each equipped with its own queue. The model can represent a distributed computing system which is composed of two loosely-coupled workstations communicating over a network.

We study a probabilistic policy, and two policies that use an estimation of job service time, one deterministic and one adaptive. The results are obtained using simulation techniques.

In the probabilistic case the scheduling policy is described by state independent branching probabilities. The job dispatcher chooses one of the two processors based on the outcome of an independent trial in which the i^{th} outcome has probability p_i.

In the deterministic case the routing decision is based on the system state. An arriving job at the proc-

essing unit is assigned by the scheduler to the processor which is expected to provide the shortest response time.

In the adaptive case an arriving job is dispatched randomly to one of the two queues with equal probability. When a processor becomes idle and there are jobs waiting at the other processor queue, then a job migrates from that queue to the idle processor. This policy is triggered when a job departs and the processor becomes idle. We have considered migration of non executing jobs. We assume that a job being executed is not eligible for transfer because of the complexity of the mechanism required in order to migrate the job in execution. The obvious disadvantage of preemptive migration is the need to transfer the memory associated with the migrant process; thus, the migration cost for an active process is much greater than the cost of remote execution.

Scheduling algorithms involve overhead which is attributed to two factors: (1) the scheduling algorithm's execution time and (2) the number of task migrations the scheduler produces.

In this paper we assume that the overhead for the scheduling algorithms is negligible. This is a reasonable assumption, given the simplicity of the algorithms and the small number of processors considered here. We assume though that when a job is transferred for remote processing, the job incurs a communication cost.

In our system after processor service, resequencing of jobs takes place which ensures that jobs leave the processing unit on a first-in-first-out basis. Jobs who arrive at the system should depart from the system in the same order as their arrivals. Therefore, after the completion of their services, jobs who go out of order are forced to wait in a special buffer—the so-called resequencing buffer—in order to rearrange their sequences, i.e. until the overtaken job of the same class completes its service. The delay due to resequencing is called resequencing delay.

The resequencing delay in multiserver queues has been studied by many authors as [4], [11], [13] and [14]. Load balancing in a system of two queues with resequencing is studied by [5]. All these works study open queuing networks and exponential CPU service times.

Load balancing with resequencing in a distributed system is studied in [8]. In that work, a probabilistic policy and the "joint the shortest queue" policy are compared. It is considered that once an arriving job is routed by the job dispatcher to one of the parallel processors it cannot be reassigned and must be processed to completion by that processor. A closed queuing network model is considered in that work and estimation of job execution times is not used.

Load balancing with task transfer is studied in [10]. Receiver-initiated and sender-initiated load sharing is studied in [2]. In [1] the task of scheduling dynamic applications that consist of single process tasks on a non-shared memory multicomputer is examined. Each task of the application is assumed to (1) require execution on a single processor, (2) have an estimate of its maximum execution time, and (3) not wait on communication with other tasks. In no one of these works resequencing is required. However, the resequencing problem can be found in several system contexts such as distributing computing systems and computer communication systems.

Sender-initiated and receiver-initiated adaptive load sharing with resequencing is studied in [6] and [7]. Transfer of non executing jobs is considered and the transfer decision is based on a threshold value of the queue lengths. No a priori information about job execution times is assumed.

In [3] a measured distribution of process lifetimes and a trace-driven simulation are used to investigate the benefits of preemptive migration.

Generally there have been numereous scheduling disciplines proposed for distributed systems, the evaluation of which has, for the most part, been based on workloads having a relatively low variability in the processing requirements of jobs. However, high performance computing centers have reported that the coefficient of variation of service times can in fact be greater than 1. Most of the models which have been studied are open and resequencing is not required.

In this work our intention is to examine the impact of job resequencing on the performance of three scheduling techniques for various coefficients of variation of the processor service times and for different degrees of multiprogramming. The first scheduling technique is a probabilistic which we use for comparison purposes. We examine if the other two techniques, which have an estimate of information about job service times, result in shorter response time at the cost of increased resequencing delay and to which extent, depending on system parameters.

This paper is theoretical in the sense that the results are obtained from simulation studies instead of from measurements of real systems. Nevertheless, we believe that the results we present are of practical value. All three algorithms we study are practical in that they are simple. Although we do not derive absolute performance predictions for specific systems and workloads, we study the relative performance of the three algorithms across a broad range of workloads and analyze how changes in the workload affect performance.

2. Model Description

We consider a queuing network model of a homogeneous distributed system. It consists of the processing unit and the I/O channel. The processing unit consists of two independent processors each serving its own queue (memory). Distributed memory systems in which processor/memory pairs are linked by an interconnection network are scalable since all requests need not use a single bus.

Since we are interested in a system with balanced program flow, we have considered an I/O channel which has the same service capacity as the processing unit.

The model is closed as the degree of multiprogramming N is constant during the simulation experiment. Neither arrivals nor departures are permitted while the system is under observation. The configuration of the model is shown in Figure 1.

In this work, jobs examined are highly independent. Specifically, once a job commences execution not a job ever idly waits for communication from another job, nor employs synchronization to ensure its concurrent execution with another job. An estimate of job execution time is known.

Each processor in the system contains a local scheduler to maintain local scheduling information and to control the local job multiplexor. The local multiplexor executes jobs in a First-Come-First-Served (FCFS) ordering. The local scheduler maintains a list of the jobs in the processor queue with associated execution times. At every moment also it can give information about the remaining processing time of the executing job and also about the total remaining processing time of queued and executing jobs.

Distributed system scheduling algorithms can be classified according to (1) the amount of information they require about jobs, and (2) the extent to which they reallocate processors among jobs.

We study a probabilistic policy, and two policies that use an estimation of job service time, one deterministic and one adaptive. We assume no overhead for the execution of the scheduling policy.

In the *Probabilistic Scheduling* (PS) case the scheduling policy is described by state independent branching probabilities. The job dispatcher chooses one of the two processors based on the outcome of an independent trial in which the i^{th} outcome has probability p_i. The p_i are assumed fixed and proportional to the processing power of processor i, i=1,2. Since one of the two processors must be chosen it holds: $\sum_{i=1}^{2} p_i = 1$ In our model the processors are homogeneous, so $p_1 = p_2 = 0.5$.

In the deterministic case, upon arrival at the processing unit, the algorithm schedules a program on the processor which will complete the program earliest given the current system state. This is the processor which has the smallest sum of execution times of the queued jobs plus the remaining time to completion of the executing job. We call this policy *Shortest Response Time Scheduling* (SRTS). The additional overhead of maintaining load information exists for this type of algorithm.

In the adaptive policy case an arriving job is dispatched randomly to one of the two queues with equal probability. When a processor becomes idle and there are jobs waiting at the other processor queue, then a job migrates from that queue to the idle processor. If the system consisted of more than two processors, then a job from the most heavily loaded processor would migrate to the idle processor. We call this policy *Migration Scheduling* (MS). We have considered migration of non executing jobs. We assume that a job being executed is not eligible for transfer. The reason for this assumption is that during mid-execution process migration both the target and originating processors incur overhead. The originating node consumes processing time when it deallocates and sends the tasks while the target node uses processing time during task reception and allocation. This makes preemptive migration consuming useful processing time.

When a job is transferred for remote processing, the job incurs a communication cost. In this model only jobs in the queue are transferred. We believe that the average transfer cost for a nonexecuting job, although nonegligible, can be expected to be quite low relative to the average cost of job processing. The communication channel is modelled as a single server queuing system, whose service time is an exponentially distributed random variable with mean α to deal with the effects of communication overhead Co. We assumed a Co of 10% of the mean job service time.

An additional assumption is that the cost to migrate a job is independent of job size. The scheduler migrates the largest job in the queue of the busy processor to the idle processor. In our model we have bound the number of task migrations by assuming that a job whose service time is less than Co is processed locally. For the MS, load information is required only when a processor becomes idle after the departure of job. However, compared with SRTS, MS has the additional overhead due to job migration.

From all three scheduling policies examined PS is the simplest as the scheduler creates a small amount of overhead consisting of a random number generation. SRTS and MS develop schedules based on global in-

formation. The collection of global information requires a non-trivial amount of overhead, but it is essential to develop a moderately effective scheduler. The schedulers operate optimally when no processing overhead is used with both formulating a schedule and migrating jobs. In this paper we assumed that the system overhead to execute a scheduling algorithm itself is considered part of job execution time.

The following abbreviations appear throughout the remainder of the paper:

PS : Probabilistic Scheduling.
SRTS : Shortest Response Time Scheduling.
MS : Migration Scheduling.

In order to implement the SRTS and MS policies, it is required that job execution times (job sizes) are known in advance. In practice, only an estimate of job execution times can be obtained. In this study we have considered an estimation error E. Job sizes estimated is assumed to be uniformly distributed within $\pm E\%$ of the exact value.

The important requirement of the model is that jobs must leave the processing unit in the order of their arrival. For this, each job enters the resequencing buffer after completion of its service, where it eventually waits until all jobs that entered the system before it have been served. We furthermore assume that the time taken by its resequencing process is negligible: if this job has no one to wait for, it leaves instantaneously the system. On the other hand, it leaves as soon as the last customer it has to wait for enters the resequencing buffer.

We also investigate the impact of the variability in job service demand on system performance. A high variability in job service demand implies that there is proportionately a high number of jobs in the system with small service demand and a comparatively low number of jobs with very large service demand. When a "large" job enters the system and is served by the processor, it will occupy the processor for a long time and depending on the scheduling policy applied it may introduce inordinate queuing delays for the other jobs waiting for service by the processor. When the MS policy is performed, jobs may be transferred during that time to the other processor.

We examine two cases of processor service time distributions:

1. Processor service times are independent and identically distributed (IID) exponential random variables with mean m at each processor.
2. Processor service times have a Branching Erlang distribution [12] with two stages and are IID. The coefficient of variation is C, where $C > 1$ and the mean is m at each processor.

With all three scheduling policies examined, the FCFS queuing discipline is assumed in each processor queue.

A job, after leaving the processing unit, requests service from the I/O unit. The I/O queuing discipline is FCFS. The I/O service times are exponentially distributed with mean k and are IID.

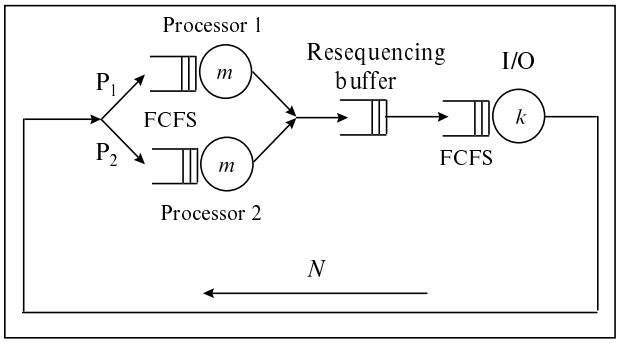

Figure 1. The queuing network model, probabilistic case.

3. Performance Parameters

We define:

Response time of a random job as the interval of time measured from an arrival of this job at the processing unit to service completion of this job (time spent in a processor queue and in service in addition to a possible communication delay incurred due to job transfer).

Cycle time of a random job as the time between two successive service requests of this job from the processing unit. So in our model cycle time is the sum of queuing and service time at a processor, communication delay, resequencing delay and queuing plus service time at the I/O unit.

We also denote the following:

RT : mean response time
D : mean resequencing delay
WSD : mean waiting and service time, communication delay and resequencing delay
K : mean cycle time
R : system throughput rate

U_{CPU} : mean processor utilization
$U_{I/O}$: utilization of the I/O channel
N : degree of multiprogramming

The external performance of our model is indicated by the mean cycle time (program performance) and the system throughput rate (system performance). The internal efficiency is primarily determined by the mean processor utilization because it represents the level of contention for the most critical resources in the system.

We use PS as a comparison measure of the performance of the SRTS and MS scheduling policies. So, when the model works first with the PS policy and then with one of the other two scheduling policies examined, we define the relative performance parameters calculated on a percentage basis as follows:

D_{RT} : relative decrease in RT
D_{WSD} : relative decrease in WSD
D_K : relative decrease in K
D_R : relative increase in R

4. Model Implementation and Input Parameters

We simulated the queuing network model with discrete event simulation models [9] using the method of independent replications. For every mean value a 95% confidence interval was evaluated. All confidence intervals were less than 5% of the mean values.

We considered a balanced system with m=1.0 and k = 0.5. We chose a mean communication delay α=0.10. The system was examined in cases of exponential processor service times (C = 1) and Branching Erlang for C = 2, 4 and 6. The degree of multiprogramming N was taken as 2, 4, 6, 8, 10.

In the SRTS and MS cases, we have considered an estimation error E= ±10%.

5. Simulation Results and Discussion

Due to space limitations, only a few results will be presented, but these will be representative of the performance of the model.

In Tables 1-3 performance parameters of the C=1 case are presented.

In Tables 4-7, relative performance parameters for the C=1 and C=2 cases are presented.

In Figures 2-9, mean cycle time and system throughput rate are plotted versus N in all cases examined.

Our results show that:

For N=2, MS performs the same as PS as there are not opportunities for job migration. SRTS for all N and MS for N>2 although yield higher resequencing delays then PS they outperform PS. This is due to the fact that SRTS and MS yield much lower response time then PS, resulting in lower WSD, lower mean cycle time and higher system throughput.

In all cases SRTS yields lower response time than MS. However, the resequencing delay is higher with the SRTS case. As a result, SRTS and MS do not differ significantly for N>2.

For all N, D_{RT} increases with increasing C. However, the superiority of SRTS and MS over PS decreases with increasing C. At C=6 all three methods perform very close. This is due to the fact that the bigger C is, the more the CPU service times vary from the mean service time m. Furthermore more service times are produced that are much shorter than m and fewer ones which are much longer than m. The very small jobs, no matter whether they are served using the PS or one of the other two scheduling policies, have a long delay at the synchronisation buffer when a large task is processed. During that time it is most probable for the I/O unit to be idle, but afterwards it is supplied with many tasks which are forced to delay in its queue.

For all C, from low to moderate loads, D_{RT} increases with increasing N. This is due to the fact that at moderate loads there are more opportunities to exploit the SRTS and MS abilities than at low loads. However, from moderate to high loads, D_{RT} values differ less significantly with increasing N as most of the time processors are busy with the PS policy.

The largest values of D_K and D_R were at C=1 and N=4 with the SRTS policy. In this case, SRTS performed about 10% better than PS. With PS mean processor utilization (U_{CPU}) was 0.62, whilst SRTS increased U_{CPU} to 0.68.

6. Conclusions and Further Research

In this work we studied the assignment of independent programs in a distributed system in conjunction with resequencing. We used simulation as the means of obtaining results.

Different scheduling policies were studied and compared for various degrees of multiprogramming N and coefficients of variation C of job execution times. The simulation results reveal the following:

- From the three policies the PS performed worst.
- For N=2 and for all C, MS performs the same as PS. In all other cases SRTS and MS do not differ significantly.

- The advantages of SRTS and MS over PS are not completely exploited due to the resequencing delay.
- The superiority of SRTS and MS over PS decreases with increasing C. The greatest difference in performance was achieved with SRTS at N=4 and C=1, where SRTS performs about 10% better than PS.

In the cases where SRTS and MS offer performance advantages over PS it should be taken into account that this is obtained at the cost of an increased overhead, such as load information management, load information distribution, transfer decision making, and task transfer delays. This overhead can seriously impact system performance when considering large systems. In this work we do not model it as the number of processors is small. As a future research we plan to consider a larger system and to count the cost of scheduling associated overhead. We expect that at large systems and at high C the PS policy should be used.

Table 1. C=1, PS case

N	U_{CPU}	$U_{I/O}$	RT	D	WSD	K	R
2	0.466	0.469	1.275	0.145	1.419	2.134	0.937
4	0.620	0.624	1.828	0.265	2.093	3.205	1.248
6	0.711	0.717	2.396	0.329	2.724	4.194	1.431
8	0.767	0.771	2.995	0.363	3.358	5.186	1.543
10	0.808	0.813	3.602	0.389	3.991	6.152	1.625

Table 2. C=1, SRTS case

N	U_{CPU}	$U_{I/O}$	RT	D	WSD	K	R
2	0.499	0.502	0.994	0.249	1.243	1.991	1.004
4	0.683	0.688	1.273	0.375	1.649	2.910	1.375
6	0.781	0.786	1.631	0.427	2.059	3.817	1.572
8	0.842	0.848	2.029	0.449	2.477	4.719	1.695
10	0.877	0.882	2.490	0.445	2.935	5.671	1.763

Table 3. C=1, MS case

N	U_{CPU}	$U_{I/O}$	RT	D	WSD	K	R
2	0.466	0.469	1.275	0.145	1.420	2.134	0.937
4	0.654	0.658	1.537	0.314	1.851	3.041	1.315
6	0.761	0.771	1.829	0.372	2.201	3.892	1.542
8	0.826	0.836	2.250	0.370	2.620	4.783	1.673
10	0.865	0.874	2.690	0.396	3.086	5.720	1.748

Table 4. C=1, PS versus SRT

N	D_{RT}	D_{WSD}	D_K	D_R
2	22.01	12.46	6.69	7.17
4	30.36	21.23	9.19	10.13
6	31.91	24.43	8.97	9.86
8	32.26	26.23	9.00	9.89
10	30.85	26.44	7.83	8.49

Table 5. C=1, PS versus MS

N	D_{RT}	D_{WSD}	D_K	D_R
2	0.00	0.00	0.00	0.00
4	15.93	11.55	5.11	5.38
6	23.66	19.22	7.20	7.76
8	24.85	21.96	7.77	8.43
10	25.30	22.66	7.02	7.55

Table 6. C=2, PS versus SRTS

N	D_{RT}	D_{WSD}	D_K	D_R
2	23.64	7.98	4.20	4.39
4	37.99	14.03	5.70	6.05
6	42.75	17.03	5.28	5.58
8	45.11	21.21	5.75	6.10
10	45.78	23.46	5.53	5.85

Table 7. *C*=2, PS versus MS

N	D_{RT}	D_{WSD}	D_K	D_R
2	0.00	0.00	0.00	0.00
4	18.47	7.04	2.97	3.06
6	27.52	13.30	4.94	5.19
8	29.89	17.44	5.77	6.12
10	34.35	21.31	6.28	6.70

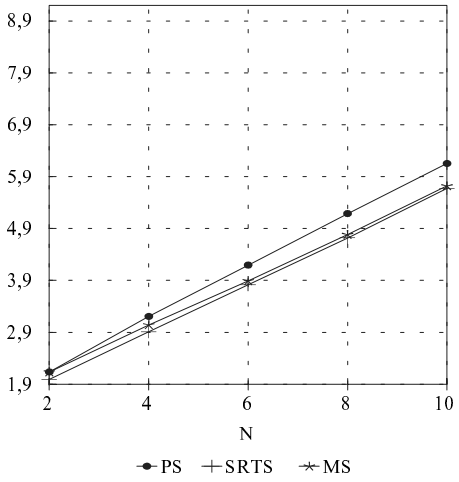

Figure 2. *K* versus *N*, *C*=1

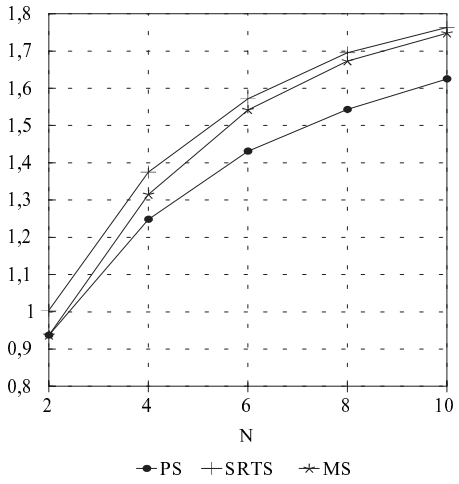

Figure 3. *R* versus *N*, *C*=1

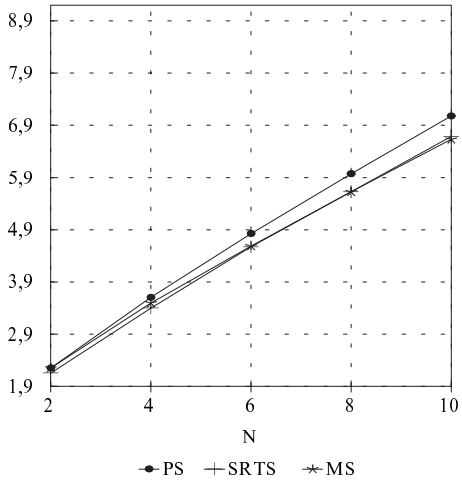

Figure 4. *K* versus *N*, *C*=2

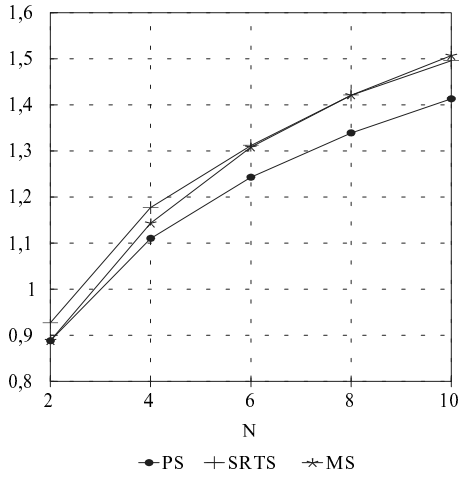

Figure 5. *R* versus *N*, *C*=2

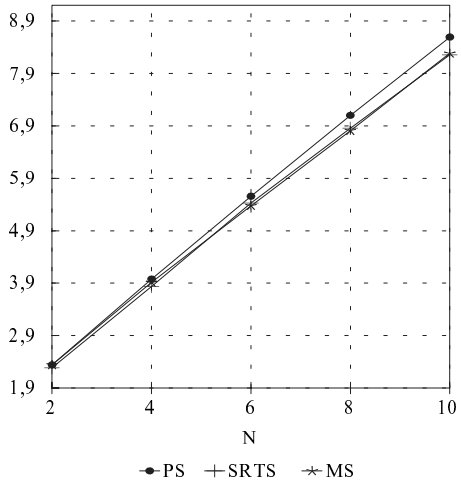

Figure 6. *K* versus *N*, *C*=4

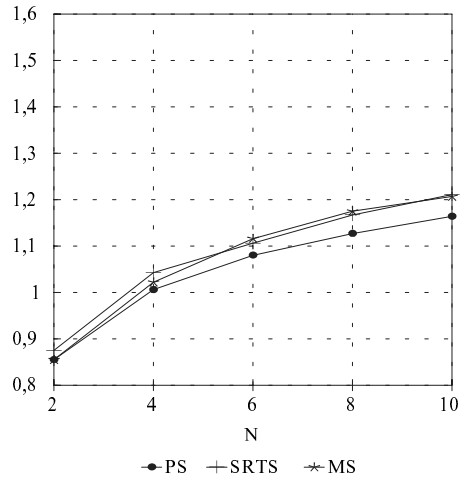

Figure 7. *R* versus *N*, *C*=4

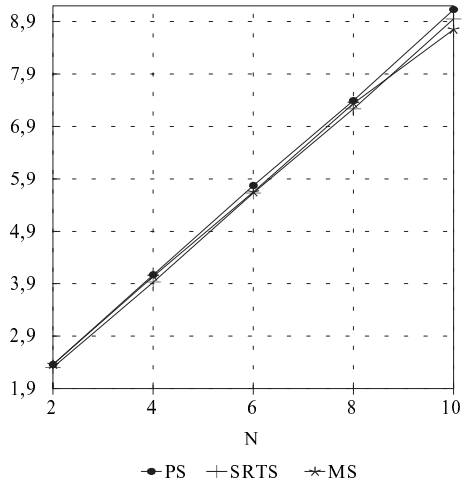

Figure 8. *K* versus *N*, *C*=6

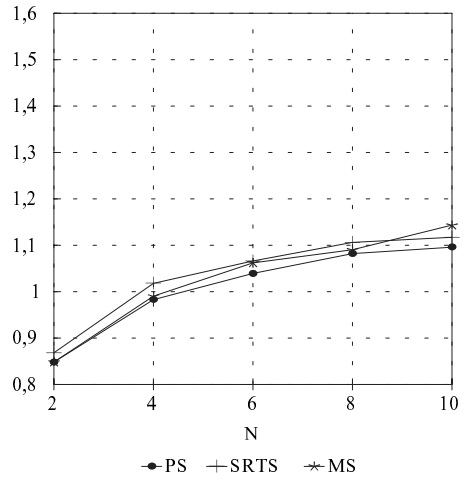

Figure 9. *R* versus *N*, *C*=6

References

[1] B.A. Blake, "Assignment of Independent Tasks to Minimize Completion Time", *Software-Practice and Experience*, Vol.22(9), 1992, pp. 723-734.

[2] D.L. Eager, E.D. Lazowska, and J. Zahorjan, "A Comparison of Receiver-Initiated and Sender-Initiated Adaptive Load Sharing", *Performance Evaluation*, Vol.6, 1986, pp. 53-68.

[3] M. Harchol-Balter, and A.B. Downey, "Exploiting Process Lifetime Distribution for Dynamic Load Balancing", In *Proceedings of Sigmetrics '96*, Philadelphia, May 23-26, 1996, pp. 13-24.

[4] I. Iliadis, and Y.C. Lien, "Resequencing delay distribution for a queuing system with two heterogeneous servers under threshold type scheduling", In *Data Communication Systems and Their Performance*, Elsevier Science Publishers B.V., IFIP, 1988, pp. 359-373.

[5] A. Jean-Marie, "Load Balancing in a System of Two Queues with resequencing", In *Performance '87*, P.-J. Courtois and G. Latouche (editors), Elsevier Science Publishers B.V. (North-Holland), 1988, pp. 75-88.

[6] H.D. Karatza, "Simulation Study of Sender-Initiated Load Sharing with Resequencing", In *Proceedings of Summer Computer Simulation Conference, SCSC'96*, SCS, V.W. Ingalls, J. Cynamon, and A.V. Saylor (editors), Portland, Oregon, 21-25 July 1996, pp. 497-501.

[7] H.D. Karatza, "Sender-Initiated versus Receiver-Initiated Adaptive Load Sharing with Resequencing". In *Proceedings of the 8th European Simulation Symposium & Exhibition*, SCS, A. Bruzzone and E. Kerckhoffs (editors), Genoa, Italy, October 24-26, 1996, pp. 546-550.

[8] H.D. Karatza, and R.C. Huntsinger, "Load Balancing and Resequencing in a Homogeneous Distributed System". In *Proceedings of Summer Computer Simulation Conference, SCSC'95*, SCS, T. Orren and L.G. Birta (editors), Ottawa, Ontario, Canada, 24-26 July 1995, pp. 790-794.

[9] A. Law, and D. Kelton, *Simulation Modeling and Analysis*, McGraw-Hill, New York, 1991.

[10] R.-C. Liu, "Analysis of the Effects of System Parameters on Load Balancing", *Information Sciences*, Vol.81, 1994, pp. 37-54.

[11] I. Sasase, and S. Mori, "Resequencing delay for a queuing system with multiple servers under threshold-type scheduling", In *Proceedings of the Tenth Annual Joint Conference of the IEEE Computer and Communications Societies*, IEEE, Vol.1, 1991, pp. 391-399.

[12] C.H. Sauer, and K.M. Chandy, *Computer Systems Performance Modelling*, Prentice-Hall, Englewood Cliffs, N.J., 1981.

[13] T. Takine, and T. Hasegawa, "Resequencing delay in Preemptive Priority M/M/2 Queues", *In Performance '90*, Elsevier Science Publishers B.V., 1990, pp. 109-121.

[14] S. Varma, "Optimal Allocation of Customers in a Two Server Queue with Resequencing", *IEEE Transactions on Automatic Control*, Vol.36(11), 1991, pp. 1288-1293.