Scheduling Strategies for Multitasking in a Distributed System

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

Abstract

In this paper we study various policies for scheduling tasks in distributed processor queues. Their performance is studied and compared for a variety of workloads. It is our intention to find a policy that increases throughput and also is fair to jobs. Simulation results indicate that the policy that schedules the shortest task in a queue, when there is not any other task that has been waiting more than some configurable period of time, yields good system performance and also provides a guarantee for fairness in individual job execution.

1. Introduction

In distributed systems, task scheduling has been shown to be a critical factor in achieving efficient parallel execution. The design of policies that schedule parallel job tasks in a manner that tends to minimize response time and maximize throughput is an important issue. And indeed, there have been studied and implemented many scheduling policies ([2], [3], [4], [5], [9]).

Distributed systems offer considerable computational power, which can be used to solve problems with large computational requirements. However, it is not always possible to efficiently execute the parallel jobs. It is critical how to partition the program into tasks, assigning the tasks to processors and scheduling task execution on each distributed processor. Good scheduling policies are needed to improve system performance while preserving individual application performance so that there will not be some jobs suffering unbounded delays.

The primary purpose in most of the existing works is to find ways to distribute the tasks among the processors in order to achieve some performance goals such as minimizing job execution time, minimizing communication and other overhead and/or maximizing resource utilization. There are cases though, where task sequencing should be preserved as much as possible so as to achieve fairness in job execution. A task that is given a low priority accordingly to the scheduling method's criteria should not be overtaken by an arbitrary number of higher priority tasks.

Dandamudi ([2], [3]) has conducted an extensive and thorough study on task scheduling for multiprocessor systems. These works examine many cases in detail. Their results indicate that scheduling policies have substantial impact on performance when non-adaptive routing strategies are used. They consider an open queuing network model and they assume that the number of tasks per job is exponentially distributed.

Kumar and Shorey ([5]) study a parallel processing system comprising several homogenous computers interconnected by a communication network. Jobs consist of a series of forks and joins. Each fork of a job gives rise to a random number of tasks that can be processed independently on any of the computers.

Sevcik [9] uses Least Work First (LWF) and Least Remaining Work First (LRWF). The results of his study confirm the value of using application characteristics in scheduling when they can be obtained.

Generally, scheduling policies for distributed systems and multiprogrammed parallel systems study the influence of the scheduling policy on processor performance only. They do not explicitly model the I/O processing, although it can significantly influence the overall system performance. However, scheduling is not an isolated issue. It is but one service provided by the operating system. The solution to the scheduling problem must be integrated with solutions to other problems, e.g. I/O management. The different parts of the system must work together to create a cohesive whole in a way that makes sense. Rosti et al. [7] study large-scale parallel computer systems and they suggest that the overlapping of the I/O demands of some jobs with the computation demands of other jobs offer a potential improvement in performance.

Numerous scheduling disciplines have been proposed for multiprocessor systems, the evaluation of which, for the most part, was conducted on workloads with a

relatively small variability in task processing requirements. However, high performance computer centers have report that their service time coefficient of variation can in fact be greater than one.

In this work we consider that a parallel program has a simple fork-join structure. Although the job structure is simple, the splitting of jobs into tasks and the resultant parallelism makes the modelling and analysis of multiprocessing systems very complicated. We study task scheduling in a closed queuing network model of a distributed system with P=16 processors where we incorporated I/O equipment. Jobs are partitioned into tasks that can be run in parallel. Each task is assigned randomly to one of the P queues with equal probability. We bound the number of tasks per job by the number of processors in the system. We consider workloads with no correlation between total service demand and number of tasks in a job.

Karatza ([4]) studied a similar model. A distributed system with the P=4 processors and an I/O unit was considered. Three scheduling policies were studied. The first-come-first served, shortest-task-first-served, and a policy that gives priority to the tasks of the job that has the smallest number of tasks. The best method was the shortest-task-first-served method. However, in that work, we studied the overall performance only, and we did not take into account fairness across competing jobs.

In this work we study the following five scheduling techniques: shortest-task-first-served, two variations of the shortest-task-first-served method and a policy that gives priority to the tasks of the job that has the smallest number of uncompleted tasks. We also examine the strict first- come-first-served policy. Our goal is to achieve high system performance in conjunction with fairness in job execution.

We compare the performance of the different scheduling policies for various coefficients of variation of the processor service times and for different degrees of multiprogramming. To our knowledge, such an analysis of task scheduling has not appeared in the research literature.

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 algorithms we study are practical in that they can be implemented. Although we do not derive absolute performance predictions for specific systems and workloads, we study the relative performance of the different task scheduling algorithms across a broad range of workloads and analyze how changes in the workload affect performance.

For simple idealised systems, performance models can be mathematically analysed using queuing theory to obtain performance measures. Our system, in addition to exponential distribution for task execution time, involves Branching Erlang distribution as well. Also, it involves fork-join programs and scheduling policies with different complexities.

For complex systems analytical modelling typically requires additional simplifying assumptions, and such assumptions might have unforeseeable influences on the results. Therefore, research efforts have been devoted to finding approximate analysis, to developing tractable models in special cases, and to conducting simulations.

Exact analysis of fork-join queuing models is well known to be intractable. For example, Kumar and Shorey ([5]) derived upper and lower bounds for the mean response time when jobs have a linear fork-join structure. In our work we chose simulations because it is possible to simulate the system under study 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 the paper is as follows. Section 2.1 specifies system and workload models, section 2.2 describes the task scheduling policies and section 2.3 presents the metrics employed in assessing the performance of the scheduling policies we study. 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. Finally the last section concludes with several comments and gives directions for further research.

2. Model and methodology

2.1 System and workload models

A closed queuing network model of a distributed system is considered. There are P homogeneous and independent processors each serving its own queue. We have examined the system for P = 16. This is a reasonable choice for the current existing medium-scale departmental networks of workstations. It is believed that qualitative result for other numbers of processors even for large-scale distributed, are similar for the data demonstrated here.

A high-speed network with negligible communication delays interconnects the distributed nodes.

The effects of the memory requirements and the communication latencies are not represented explicitly in the system model. Instead, they appear implicitly in the shape of the job execution time functions. By covering several different types of job execution behaviours, we expect that various architectural characteristics be captured, as well.

The degree of multiprogramming N is constant during the simulation experiment. A fixed number of jobs N is circulating alternatively between the processors and the I/O unit.

The configuration of the model is shown in Figure 1.

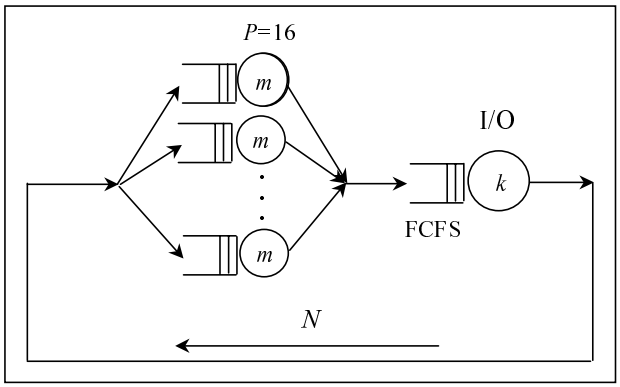

Figure 1. The queuing network model

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 processors.

An important part of a distributed system design is the 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.

In this work jobs are partitioned into tasks that can be run in parallel. On completing execution, a task waits at the join point for its sibling tasks of the same job to complete execution. Therefore synchronization among tasks is required. The price that one pays for the increased parallelism is the synchronization delay that occurs when tasks wait for their siblings to finish execution.

Each task is assigned randomly to one of the queues with equal probability. Tasks in processor queues are executed according to the scheduling method that is currently employed. No migration or preemption is permitted. Once a task starts execution, then it will run to completion without interruption.

The technique used to evaluate the performance of the scheduling disciplines is experimentation using a synthetic workload simulation. In studies like this, one is usually required to use synthetic workloads because real workloads cannot be simulated efficiency enough and real systems with actual workloads are not available for experimentation. Also, useful analytic models are difficult to derive because the subtleties between various disciplines are difficult to model and because the workload model is quite complex.

The workload considered here is characterized by three parameters:

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

We assume that there is not correlation between the different parameters. For example, a job with a small number of tasks may have a long execution time.

Jobs consist of a set of $n \geq 1$ tasks. We assume that the number of tasks of jobs is uniformly distributed in the range of $[1..P]$. We have chosen the uniform distribution because it has a reasonable large deviation and therefore more facts about system behavior can be revealed. The number of tasks of a job x is represented as $t(x)$.

If $p(x)$ represents the number of processors required by job x, then the following relation holds:

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

Each time a job returns from I/O service to distributed processors, it is partitioned into a different number of tasks and it needs a different number of processors for execution. That is its degree of parallelism is not constant during its lifetime in the system.

We also investigate the impact of the variability in task service demand on system performance. A high variability in task service demand implies that there is proportionately a high number of service demands that are very small compared to the mean service time and a comparatively low number of service demands that are very large. When a task with a long service demand starts execution, it occupies a processor for a long interval of time and depending on the scheduling policy applied it may introduce inordinate queuing delays for the other tasks waiting for service.

The parameter, which represents the variability in task execution time, is the coefficient of variation of execution time (C). This is the ratio of the standard deviation of task execution time to its mean.

We examine the following cases:

- Task execution times are independent and identically distributed (IID) exponential random variables with mean m.
- Task execution times have a Branching Erlang distribution ([1], [8]) with two stages and are IID. The coefficient of variation is C, where $C > 1$ and the mean is m.

After a job leaves the processors, it requests service on 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.

All notations used in this paper appear in Table 1.

2.2 Job scheduling policies

The performance of programs that consist of parallel tasks is significantly affected by the choice of the policy used to schedule tasks.

The main objective of the processor schedulers is to achieve high overall system throughput while at the same time providing some guarantee for the performance of individual jobs in the system.

In this work we examine only non-preemptive task scheduling policies. We assume that the scheduler has perfect information when making decisions, i.e. it knows:

- The execution time of tasks.
- The period of time a job is in the processors system.
- The number of uncompleted tasks of each job.

Next we describe the scheduling strategies employed in this work. As in most studies we assume that their overhead is negligible.

- *First-Come-First-Served* (*FCFS*)*:* With this strategy, each task is scheduled into the assigned queue in the order of its arrival. This policy is the simplest form of task scheduling.
- *Shortest Task 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. However, it should be noted that a priori information is not often available and only an estimate of task execution time can be obtained. In this study, task execution time estimated is assumed to be uniformly distributed within $\pm E\%$ of the exact value.
  It is obvious that this method is not fair in the case where there is a task in a queue with very large service demand as it may be scheduled after a very long wait in the queue. This problem is eliminated by using the following two methods that are versions of the STF method.
- *STF-L*: With this policy tasks are placed in the queue in the order they arrive, and then the STF policy is applied to the first L tasks in the queue.
- *STF-Maximum Wait* (*STF-MW*): With this scheduling scheme, priority is given to tasks of jobs that are in the system for more than a configurable period of time *MW*. Otherwise the STF policy is applied.
- *Job with the Smallest Number of Uncompleted Tasks First* (*JSNUTF*): This policy gives higher priority to tasks that belong to the job with the smallest number of uncompleted tasks. This number is an indication of how close is a job to completion. This method does not need information about task execution time but it is obvious that incurs an additional overhead, as the scheduler has to know at any moment the number of uncompleted tasks of all jobs in all queues. Also it has to find the smallest of these numbers every time a task departs a processor after service completion.

When using priorities, in the case of ties the *FCFS* method is used.

2.3 Performance metrics

We consider the following definitions:

- *Response time* of a random job is the interval of time from the dispatching of this job tasks to processor queues to service completion of the last task of this job. We use the *Maximum Response Time* observed during a simulation experiment as the indication of fairness.
- *Cycle time* of a random job is the time that elapses between two successive processor service requests of this job. In our model cycle time is the sum of response time plus queuing and service time at the I/O unit.

Parameters used in later simulation computations are presented in Table 1.

Table 1. Notations

RT	Mean response time
RT_{max}	Maximum response time
K	Mean cycle time
R	System throughput
U_P	Mean processor utilization
N	Degree of multiprogramming
C	Coefficient of variation
m	Mean task execution time
k	Mean I/O service time
E	Estimation error in task execution time
L	Number of the first positions in a queue that the STF-L policy examines
MW	Maximum wait time that is the threshold for the STF-MW policy

3. Simulation results and discussion

3.1 Model implementation and input parameters

The queuing network model was simulated with discrete event simulation models ([6]) using the independent replication method. For every mean value a 95% confidence interval was evaluated. All confidence intervals were found to be less than 5% of the mean values. The system considered is balanced:

$$m=1.0, \quad k = 0.531$$

The reason we have chosen k = 0.531 for balanced program flow is that at the processors there are on average 8.5 tasks per job. So, when all processors are busy, an average of 1.882 jobs is served each unit of time. This implies that I/O mean service time must be equal to 1/1.882 = 0.531 if the I/O unit is to have the same service capacity.

The system was examined in cases of task execution time with exponential distribution (C = 1), and Branching Erlang for C = 2, 4.

The degree of multiprogramming N was taken as 8, 12, 16, 20, and 24. The reason we have examined various numbers of programs N is because this is a critical parameter, which reflects system load.

In the STF, STF-L and STF-MW cases we have also examined estimation errors ±10% and ±30%.

In this work we have considered L=N/2, N/4. This is a reasonable assumption because the maximum queue lengths with the STF method were around N. In the STF-MW case we examined MW=2*N, N, N/2, N/4.

3.2 Performance analysis

Due to space limitations, only few results are presented, but they are representative of the overall model performance (Tables 2-7, Figures 2-11).

Simulation results show the following:

In all cases examined, the performance in terms of mean cycle time and system throughput is superior with the STF method. However, this policy presented larger values of RT_{max} then those presented by the other methods. FCFS performed worst than all methods, but it produced small RT_{max}.

From the other methods, the smallest values of RT_{max} were produced by the STF-MW case for all examined values of the parameter MW. For C=1, and MW=2*N, this method performed very close to STF. As MW decreases, the performance of the method deteriorates but the RT_{max} decreases. This is because while MW decreases this method tends to favour the first tasks in the queues and therefore it behaves similarly to the FCFS method.

In the MW=2*N case STF-MW improves the overall performance by up to 19% over FCFS while maintaining fairness. For MW=N the improvement is smaller but the values of RT_{max} produced by these two methods are in the same range. For all N and MW, the performance of STF-MW decreases with increasing C. For MW=2*N, for all C the decrease is not serious, and STF-MW performs well relatively to FCFS. However, for MW=N/4 the performance of STF-MW degrades seriously with increasing C. For C=2, 4 the STF-MW and FCFS policies perform almost the same.

The STF-L policy for L=N/2 performed very close to STF for all C and N. For L=N/4, due to the small number of tasks in the queues that this method examines, its performance deteriorates compared with the performance of STF and JSNUTF. However, the difference in performance is smaller at higher C. For C=4, these three methods perform almost the same.

JSNUTF performed very close to STF. Both, JSNUTF and STF-L produced RT_{max} much larger than STF-MW produced for all MW.

We conducted additional simulation experiments to assess the impact of service time estimation error on the performance of the STF, STF-L and STF-MW methods. Figure 11 shows the effect of service time estimation error on system throughput for the STF-MW, MW=N, C=1 case. The estimation error in this figure is set at ±0%, ±10% and ±30%. The graph shows that the estimation error in task service time affects marginally system performance. Therefore, no profit can be gained from the a priori knowledge of exact task service times.

All five policies have their merits. STF, STF-L and STF-MW methods assume a priori knowledge of an approximate task execution time. JSNUTF assumes a global information about the tasks of all jobs in the system. On the other head FCFS is easier to implement and it results in less overhead. Therefore we come to the conclusion that the STF-MW is a good policy to choose when it is feasible to implement it in practice.

4. Conclusions and further research

In this work we studied task scheduling in a distributed system.

Our objective was to obtain good overall system performance while at the same time maintaining fairness in individual job execution time. We used simulation as the means of obtaining results.

Five scheduling policies were considered: FCFS, STF, STF-L, STF-MW, and JSNUTF. Their performance was studied and compared for various degrees of multiprogramming N and coefficients of variation C of task execution times.

The simulation results reveal the following:

- In all cases examined the STF-MW method for $MW=2*N$ provided both, good system performance and fairness in individual job execution.
- STF-MW performed better for $C=1$ as it performed very close to STF. For $C>1$ the overall system performance when using the STF-MW policy decreases with increasing C but remains constantly better than when using FCFS. Further to this, STF-MW is the fairest from the rest of the other policies examined.

This work is a case study. It should be extended to the following directions:

- Different job size distributions to be considered.
- Pre-emptive scheduling methods to be studied.

Table 2. *C*=1, FCFS case

N	U_P	RT	K	R	RT_{max}
8	0.58	6.16	7.26	1.10	23.75
12	0.67	8.00	9.44	1.27	32.15
16	0.73	9.83	11.65	1.37	38.76
20	0.77	11.56	13.76	1.45	42.58
24	0.80	13.35	15.86	1.51	49.59

Table 3. *C*=1, STF case

N	U_P	RT	K	R	RT_{max}
8	0.61	5.07	6.29	1.27	69.62
12	0.70	6.17	7.98	1.50	101.20
16	0.76	7.13	9.69	1.65	154.90
20	0.79	8.09	11.51	1.74	193.80
24	0.81	8.71	13.33	1.80	216.10

Table 4. *C*=1, STF-L case, *L=N/2*

N	U_P	RT	K	R	RT_{max}
8	0.60	5.25	6.44	1.24	54.47
12	0.70	6.26	8.04	1.49	76.73
16	0.75	7.26	9.81	1.63	150.74
20	0.78	8.1	11.56	1.73	190.03
24	0.81	8.86	13.38	1.79	192.44

Table 5. *C*=1, STF-MW, *MW=2*N*

N	U_P	RT	K	R	RT_{max}
8	0.60	5.12	6.34	1.26	34.25
12	0.70	6.19	8.02	1.50	40.53
16	0.75	7.24	9.76	1.64	47.70
20	0.78	8.12	11.57	1.73	57.23
24	0.81	9.038	13.44	1.79	65.139

Table 6. *C*=1, STF-MW, *MW=N*

N	U_P	RT	K	R	RT_{max}
8	0.59	5.34	6.52	1.23	23.26
12	0.68	6.50	8.27	1.45	29.97
16	0.73	7.73	10.1	1.58	37.55
20	0.77	8.64	11.90	1.68	43.06
24	0.79	9.70	13.88	1.73	47.16

Table 7. *C*=1, JSNUTF case

N	U_P	RT	K	R	RT_{max}
8	0.61	5.19	6.41	1.25	43.34
12	0.70	6.32	8.08	1.48	53.99
16	0.76	7.30	9.76	1.64	72.87
20	0.79	8.08	11.49	1.74	117.26
24	0.81	8.86	13.32	1.80	194.22

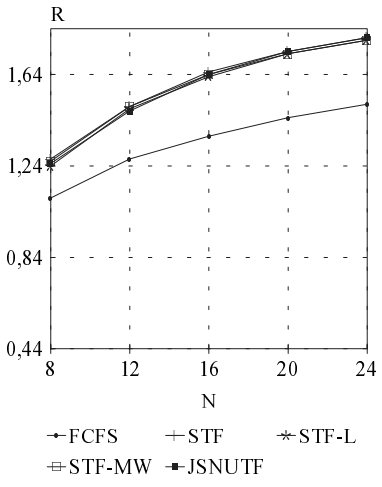

Figure 2. *R* versus *N*, C=1 (*L=N/2*, *MW = 2*N*)

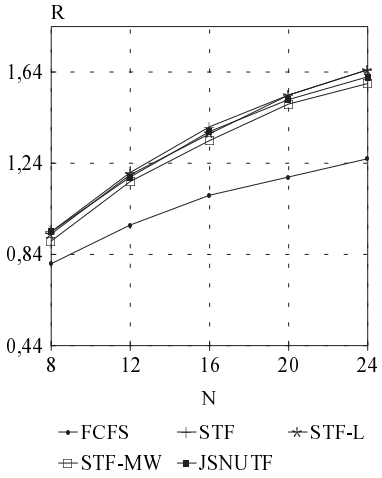

Figure 3. *R* versus *N*, *C*=2 (*L*=*N*/2, *MW* = 2N*)**

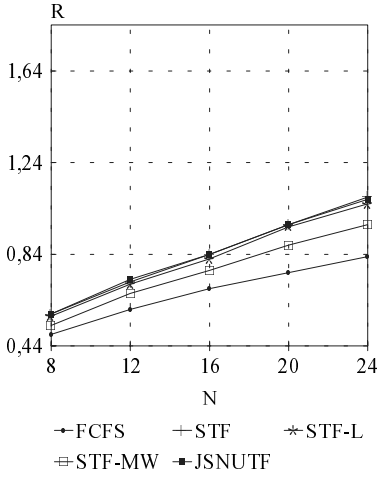

Figure 4. *R* versus *N*, *C*=4 (*L*=*N*/2, *MW* = 2N*)**

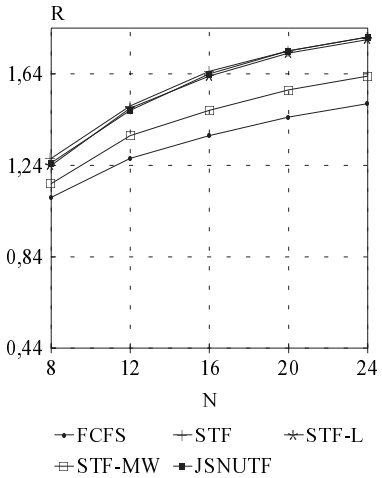

Figure 5. *R* versus *N*, *C*=1 (*L*, *MW* = *N* / 2)

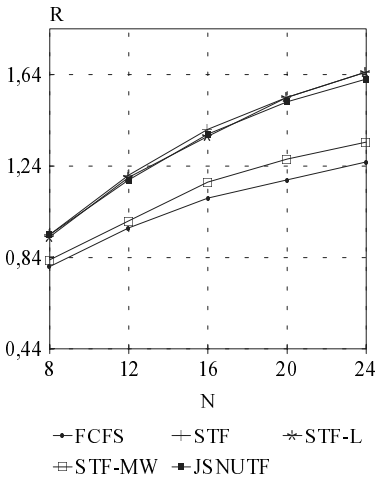

Figure 6. *R* versus *N*, *C*=2 (*L*, *MW* = *N* / 2)

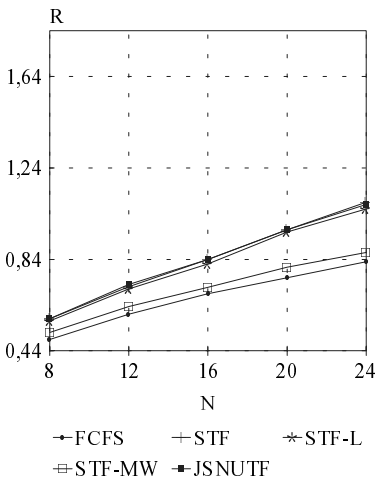

Figure 7. *R* versus *N*, *C*=4 (*L*, *MW* = *N* / 2)

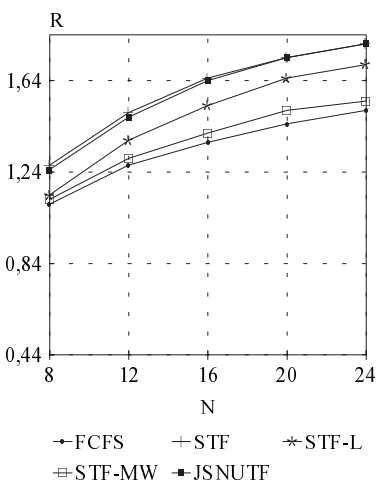

Figure 8. *R* versus *N*, *C*=1 (*L*, *MW* = *N* / 4)

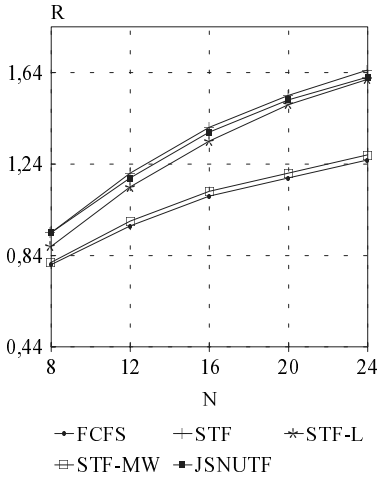

Figure 9. *R* versus *N*, *C*=2 (*L*, *MW* = *N* / 4)

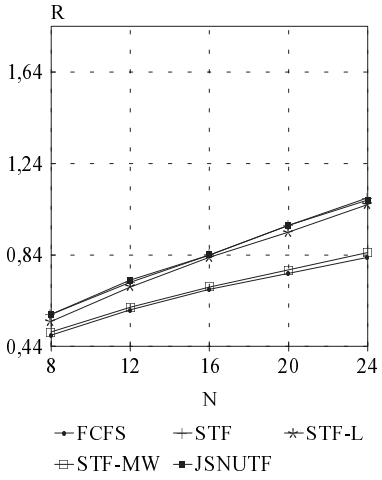

Figure 10. *R* versus *N*, *C*=4 (*L*, *MW* = *N* /4)

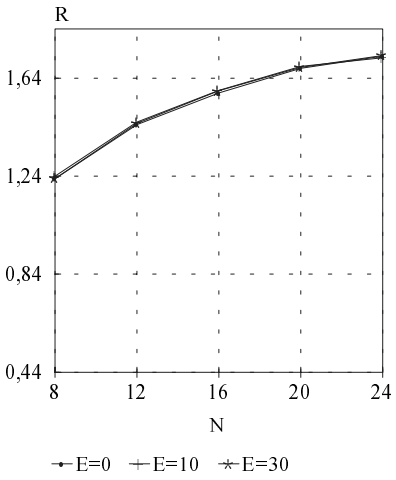

Figure 11. *R* versus *N*, *C*=1, STF-MW, *MW*=*N*

References

[1] Bolch, G., S. Greiner, H. de Meer, and K. S. Trivedi, *Queueing Networks and Markov Chains*, J. Wiley & Sons Inc., New York, 1998.

[2] S. Dandamudi, "A Comparison of Task Scheduling Strategies for Multiprocessor Systems", *Proceedings of the IEEE Symposium on Parallel and Distributed Processing*, IEEE, Dallas, TX, Dec. 1991, pp. 423-426.

[3] S. Dandamundi, "Performance implications of task routing and task scheduling strategies for multiprocessor systems", *Proceedings of the IEEE-Euromicro Conference on Massively Parallel Computing Systems*, IEEE, Ischia, Italy, May 1994, pp. 348-353.

[4] H.D. Karatza, "Simulation Study of Task Scheduling and Resequencing in a Multiprocessing System", *Simulation Journal*, Special Issue: Modelling and Simulation of Computer Systems and Networks: Part Two, SCS, April 1997, pp. 241-247.

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

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

[7] E. Rosti, G. Serazzi, E. Smirni, and M. Squillante, "The Impact of I/O on Program Behaviour and Parallel Scheduling", *Proceedings of SIGMETRICS 98*, ACM, Madison, WI, June 1998, pp. 56-65.

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

[9] K. Sevcik, "Application Scheduling and Processor Allocation in Multiprogrammed Parallel Processing Systems", *Performance Evaluation*, Elsevier B.V., Vol. 19, 1994, pp. 107-140.