A COMPARATIVE ANALYSIS OF SCHEDULING POLICIES IN A DISTRIBUTED SYSTEM USING SIMULATION

HELEN D. KARATZA

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

Abstract: In this paper we study scheduling in a distributed system. A simulation model is used to address performance issues associated with scheduling. Three policies which combine processor and I/O scheduling are used to schedule parallel jobs for a variety of workloads. Fairness is required among competing jobs. Simulated results reveal that all scheduling methods have merit, but one method significantly improves the overall performance and also provides a guarantee for fairness in individual job execution.

***Keywords*:** Simulation, Distributed Systems, Scheduling, and Performance.

1 INTRODUCTION

The scheduling of jobs in distributed systems is a heavily researched activity. However, it is still not always possible to efficiently execute parallel jobs. To achieve this, it is critical to partition the program into tasks properly, assigning the tasks to processors and scheduling execution on each distributed processor. Good scheduling policies are needed to improve system performance while preserving individual application performance so that some jobs do not suffer unbounded delays.

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

Most of the research in this area has focused on the scheduling of processors only. [Dandamudi, 1994] conducted an extensive and thorough study of task scheduling in multiprocessor systems. Results from that study indicate that scheduling policies have substantial impact on performance when non-adaptive routing strategies are used. An open queuing network model is considered and it is assumed that the number of tasks per job is exponentially distributed.

[Kumar and Shorey, 1993] 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, 1994] uses Least Work First (LWF) and Least Remaining Work First (LRWF). The results of this study confirm the value of using application characteristics in scheduling when they are available.

However, improvements in processor speeds and larger main memory sizes expose I/O subsystems as a significant bottleneck, one that prevents applications from achieving full system utilization. This problem is more serious in multiprocessor systems where multiple processors share the I/O subsystem. Therefore, I/O scheduling should be examined along with parallel job scheduling. For example, [Rosti et al, 1998] study large-scale parallel computer systems and suggest that the overlapping of the I/O demands of some jobs with the computation demands of other jobs offers a potential improvement in performance.

I/O scheduling has been examined in [Kwong and Majumdar, 1999], [Seltzer et al, 1990], and [Worthington et al, 1994]. However those studies do

not consider processor scheduling too. In [Karatza, 1997; Karatza, 2000a] we study task scheduling in distributed systems but we apply only the FCFS scheduling policy at the I/O unit. We consider various scheduling methods that apply to jobs with highly variable degrees of parallelism.

I/O scheduling along with distributed processor scheduling has also been studied in [Karatza, 2000b]. However, only a special processor scheduling method called gang scheduling is applied and job tasks are highly dependent in that they need to start at the same time and execute at the same pace. Also the degree of parallelism of jobs is highly variable.

In the study presented in this paper, job tasks are highly independent and they can execute at any time and at any order. Two different cases of job parallelism are examined: in one case, jobs have highly variable degrees of parallelism during their lifetime, while in the second case, the majority of jobs exhibit a moderate parallelism. The performance of combined processor and I/O scheduling policies for various coefficients of variation of processor service times and for different degrees of multiprogramming is compared. A closed queuing network model of a distributed system is considered which incorporates I/O equipment.

Our goal is to achieve high system performance in conjunction with fairness in job execution. To our knowledge, such an analysis of combined processor and I/O scheduling does not appear in research literature for this kind of a distributed system operating with this type of workload.

This is an experimental study in the sense that the results are obtained from simulation tests instead of from measurements of real systems. Nevertheless, we believe that the results are of practical value. Although absolute performance predictions are not derived for specific systems and workloads, we study the relative performance of differing scheduling algorithms across a broad range of workloads and we analyze how changes to 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, Branching Erlang and exponential distributions are used to compute task execution time. Also, 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 the paper is as follows. Section 2.1 specifies system and workload models, section 2.2 describes 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 summarizes findings and offers suggestions 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 queue and interconnected by a high-speed network with negligible communication delays. We examine the system for $P = 16$ processors. This is reasonable for current existing medium-scale departmental networks of workstations.

The I/O subsystem may consist of an array of disks (multi-server disk center) but it is modeled as a single I/O node with a given mean service time k. We consider that each I/O request forks in sub-requests that can be served by the parallel disk servers.

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 will be captured.

The degree of multiprogramming N is constant during the simulation experiment. A fixed number

of jobs N are circulating alternatively between the processors and the I/O unit. The configuration of the model is shown in Figure 1.

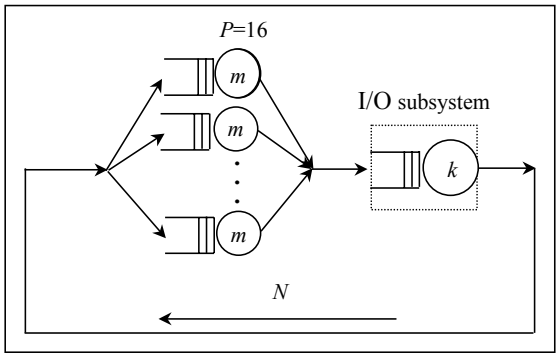

Figure 1. The queuing network model

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

An important part of the distributed system design is the workload shared among the processors and the I/O subsystem. This involves 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. It also includes the scheduling of jobs at the I/O subsystem.

Jobs are partitioned into independent tasks that can run in parallel. The number of tasks that a job consists of is this job's *degree of parallelism*. On completing execution, a task waits at the join point for sibling tasks of the same job to complete execution. Therefore, synchronization among tasks is required. The price paid for increased parallelism is a synchronization delay that occurs when tasks wait for 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 pre-emption is permitted.

The technique used to evaluate the performance of the scheduling disciplines is experimentation using a synthetic workload simulation.

The workload considered here is characterized by four parameters:

- The distribution of the number of tasks per job.
- The distribution of task execution time.
- The distribution of I/O service time.
- The degree of multiprogramming.

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.

2.1.1 Distribution of the number of tasks per job

Two different types of distribution of the number of tasks per job have been utilised:

a. Uniform distribution. We assume that the number of tasks of jobs is uniformly distributed in the range of [1..P]. Therefore, the mean number of tasks per job is equal to the $\eta = (1+P)/2$.

b. Normal distribution We assume a "bounded" normal distribution for the number of tasks per job with mean equal to $\eta = (1+P)/2$. We have chosen standard deviation $\sigma = \eta /4$.

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:

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

Each time a job returns from I/O service for scheduling on distributed processors, it is partitioned into a different number of tasks and it needs a different number of processors to execute. In other words, its degree of parallelism is not constant during its lifetime in the system. It is obvious that jobs of the uniform distribution case present larger variability in their degree of parallelism than jobs whose number of tasks is normally distributed. In the second case, most of the jobs have a moderate degree of parallelism (close to the mean η).

In this paper, in addition to the variability in jobs degree of parallelism, the impact of the variability in task service demand on system performance is also examined. 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 there are 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, it may intro-

duce inordinate queuing delays for other tasks waiting for service.

2.1.2 Service time distribution

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 with regard to task execution time distribution:

- Task execution times are independent and identically distributed (IID) exponential random variables with mean m.
- Task execution times have a Branching Erlang distribution [Bolch, 1998] 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 service times are exponentially distributed with mean k and are IID. All notations used in this paper appear in Table 1.

2.2 Scheduling Strategies

In the work presented in this paper, we assume that the scheduler has perfect information when making decisions, i.e. it knows:

- The task execution time.
- The I/O service time.
- The period of time a job is in processor queues or in the I/O queue.

We now describe the scheduling strategies employed in this work. We assume that the scheduling overhead is negligible.

For processor scheduling, the following policies are considered:

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-First **(*STF*)**. This policy assumes a-priori knowledge about a task 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 often not available and only an approximation of task execution time is available.

STF-Maximum Wait **(*STFMW*)**. With this scheduling scheme, priority is given to tasks that have been in the system for more than a configurable period of time MW. Otherwise, the STF policy is applied.

For the I/O subsystem, the following scheduling policies are examined:

First-Come-First-Served **(*FCFS*)**. This disk scheduling policy often results in suboptimal performance. Many scheduling algorithms have been proposed that achieve higher performance by taking information about individual requests into account. This policy is used as a performance yardstick against which the other policies are compared to see if job-based I/O scheduling produces any performance benefits.

Shortest-Time-First **(*STF*)**. This policy chooses the request which yields the shortest I/O time, including both seek time and the rotational latency. STF is expected to yield the best throughput since the fastest I/O service is always selected. The algorithm scans the entire queue calculating how much time each request will take. It selects the request with the shortest expected service time. It is obvious that this method is not fair when there is a job in the I/O queue with very large service demand since it may be scheduled after a very long wait in the queue. The following variation of the STF strategy eliminates this problem.

Weighted-Shortest-Time-First **(*WSTF*)**. This method is a variation of the algorithms proposed by [Seltzer et al, 1990; Worthington et al, 1994]. Priority is given to requests that have been pending in the queue for excessive periods of time. The priority may slowly increase as the request ages, or a time limit may be set after which requests are served on a FCFS basis. This algorithm applies the "standard shortest-time-first technique", but applies an ageing function to the times computed as follows:

1. First, we assume that the FCFS policy is applied to jobs that are waiting in the queue for a time interval which is greater than $10 * k$, where k is the mean I/O subsystem service time. This means that these jobs are given the highest priority, and that the expected number of other jobs, which may have bypassed them in the I/O subsystem, cannot be greater than 10.

2. For each STF calculation, the actual I/O time is multiplied by a weighting value *W*. *W* is computed by calculating how much time is left before a request will exceed the time interval 10 * *k*. Thus, the weighted time is calculated as follows. Let:

T_w be the Weighted Time
T_{real} be the actual I/O Time
T_{max} be equal to 10 * *k*
T_{wait} be the amount of time the request has been waiting for service
$T_w = T_{real} * (T_{max} - T_{wait}) / T_{max}$

In [Seltzer et al, 1990] the Maximum response time allowed (*M*) is used instead of the maximum wait time (T_{max}) that we use. In the simulations of [Seltzer et al, 1990] the time chosen for *M* is 30 seconds based on how frequently the UNIX kernel flushes its buffer cache. [Worthington et al, 1994] use the Aged Shortest-Positioning-Time-First (ASPTF) algorithm. The algorithm adjusts each positioning delay prediction (T_{pos}) by subtracting a weighted value corresponding to the amount of time the request has been waiting for service (T_{wait}). The resulting effective positioning delay (T_{eff}) is used to select the next request: $T_{eff} = T_{pos} - (W * T_{wait})$. Values of *W* range from 0 to 30.

For our study the following policy combinations were used (for processor and I/O subsystem scheduling policies respectively):

- ***FCFS-FCFS*** This is the fairest of all methods that we examine.
- ***STF-STF*** This method is not fair, but it can be applied in production systems
- ***STFMW-WSTF*** This method is intended to achieve performance close to that of the STF-STF policy, while also providing a guarantee that no job will suffer unbounded delays.

Processor and I/O estimated service times are assumed to be uniformly distributed within ± *E%* of the exact value.

2.3 Performance Metrics

The following definitions are introduced:

- *Response Time-1* of a random job is the time interval from the dispatching of a job's tasks to processor queues, to service completion at the processors of the last task of this job.
- *Response Time-2* of a random job is the queuing and service time of a job at the I/O subsystem.

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

Table 1. Notations

MRT1	Maximum Response Time-1
MRT2	Maximum Response Time-2
R	System throughput
N	Degree of multiprogramming
C	Coefficient of variation
m	Mean task execution time
k	Mean I/O service time
E	Estimation error in service time
MW	Maximum wait time that is the threshold for the STFMW policy

R represents system performance while *MRT1* and *MRT2* represent fairness of the policy that is applied. When the STF-STF or the STFMW-WSTF policies are compared with FCFS-FCFS, the relative (%) increase in *R* is represented as D_R.

3 SIMULATION RESULTS AND DISCUSSION

3.1 Model Implementation and Input Parameters

The queuing network model is simulated with discrete event simulation modelling [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. The system considered is balanced (refer to Table 1 for notations):

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

The reason $k = 0.531$ is chosen for balanced program flow is that there are on average 8.5 tasks per job at the processors. So, when all processors are busy, an average of 1.882 jobs are 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 for cases of task execution time with exponential distribution (C = 1), and Branching Erlang for C = 2, 4.

The degree of multiprogramming N is 16, 24, 32, 40, 48. The reason various numbers of programs N are examined is because it is a critical parameter that reflects the system load. In cases where estimation of service time is required, we have also examined estimation errors of ±10% and ±30%.

In the STFMW-WSTF case we examine MW=N and MW=2*N. However, the results of the MW=2*N case are analysed only because *MRT1* and *MRT2* were significantly smaller than in STF-STF case and also because the performance was significantly better than the performance of FCFS-FCFS.

3.2 Performance Analysis

A large number of simulation experiments were conducted, but to conserve space, only a subset of the experimental results is examined in this paper.

In Figures 2-7, D_R versus N is depicted while Figures 8-13 show the ratio of *MRT1* as well as the ratio of *MRT2* in each one of the STF-STF and STFMW-WSTF cases over the corresponding value in the FCFS-FCFS case. The following conclusions may be drawn by the presented results:

With regard to processor load, in the C=1 case the mean processor utilization varies approximately between 0.72 (N=16, FCFS-FCFS) and 0.92 (N=48, STF-STF). For C>1, in the corresponding cases utilization is lower. The mean I/O unit utilization varies in the same range because the system is balanced.

For all N and C, STF-STF performed better than the other two methods, while STFMW-WSTF performed better than the FCFS-FCFS method.

The relative difference in performance between policies depends on the degree of multiprogramming (system load), the variability of processor service times (coefficient of variation C), and on a job's degree of parallelism.

For C=1, the superiority of STF-STF and STFMW-WSTF over FCFS-FCFS decreases with increasing N. In the Uniform distribution case, for N=16, STF-STF is 8% better than FCFS-FCFS while STFMW-WSTF is 5% better. For N=48 all three methods have similar performance. In the Normal distribution case, D_R is a little larger. For example, for N=16, STF-STF is 9% better than FCFS-FCFS, while STFMW-WSTF is about 7% better than FCFS-FCFS.

For C=2, in the Uniform distribution case, the superiority of STF-STF and STFMW-WSTF over FCFS-FCFS increases as N increases from 16 to 24, but decreases as N continues to increase. For the same C, in the Normal distribution case, D_R increases as N increases from 16 to 40 and then it decreases with increasing N. For C=4, as N increases, D_R generally increases too.

Although D_R varies differently at different values of C as the degree of multiprogramming increases from 16 to 48, D_R changes in a similar manner at different values of C when mean processor utilization is chosen as the parameter of reference. The reason for this is because at larger values of C, larger degrees of multiprogramming should be used so that the same utilization can be achieved. For example, in the Uniform distribution case, with C=1, N=16 where mean processor utilization is 0.72, 0.78, and 0.77 in the FCFS-FCFS, STF-STF, and STFMW-WSTF cases respectively, D_R decreases with increasing N. These results are achieved at N≥24 when C=2, and at N≥ 40 when C=4. This is why as C increases, D_R starts to decrease at larger values of N than it does in the C=1 case.

In the STF-STF case, for all N, D_R increases with increasing C. This is because it is more probable at higher C to have higher variability in processor service times than at lower C. Therefore, at higher C the advantages of the STF-STF method are better exploited. In the Uniform distribution case, the highest values observed are close to 25% and they are obtained at C=4 for N=40 and 48. In the Normal distribution case, the maximum D_R observed was about 28% at C=4 and N=48.

In the STFMW-WSTF case, for all N, D_R is higher at C=2, 4 than at C=1. The highest value observed was for C=4, at N=48 at17.5% in the Uniform distribution case and 15.5% in the Normal distribution case. For some N, D_R was higher for C=2 than for C=4.

Figures 8-13 show that in all cases *MRT1* and *MRT2* are smaller in the STFMW-WSTF case than in the STF-STF case. For example, in the Uniform distribution case, for C=1, and N=40, the maximum response time at the processors is 30 times larger in the STF-STF case than in the FCFS-FCFS case. In the STFMW–WSTF case *MRT1* is 10 times larger than that of the FCFS-

FCFS case. In the Normal Distribution case for C=1, and N=48, the ratio of I/O response time is about 100 in the STF-STF case while in the STFMW-WSTF case it is only 1.3.

We also observe that *MRT1* and *MRT2* ratios generally decrease with increasing C. This is because the large variability in service times at high C produces very few large service times. In the FCFS-FCFS case, the blocking of small tasks behind a large task in the processor queue introduces queuing delays as well as synchronization delays among the sibling tasks. During this time the I/O subsystem may starve but later become deluged with jobs that must spend a large time waiting in the I/O queue.

Additional simulation experiments were conducted to assess the impact of service time estimation error on the performance of the scheduling methods that require a-priori knowledge of service times. Figure 14 shows the effect of service time estimation error on system throughput for the STFMW-STF case where C=4 in the uniform distribution case. The estimation error is set at ±0%, ±10% and ±30%. The graphs show that the estimation error in processor and I/O service time only marginally affects system performance. Therefore, no profit is gained from having exact a-priori knowledge of service times.

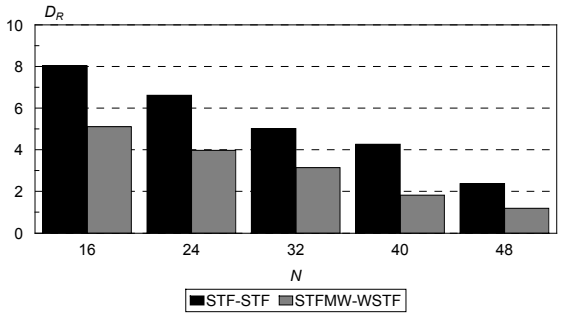

Figure 2. D_R versus N, C=1, Uniform distribution case

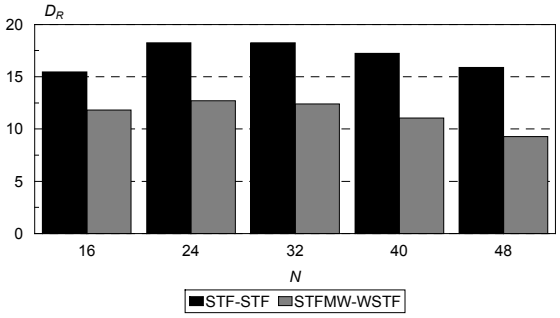

Figure 3. D_R versus N, C=2, Uniform distribution case

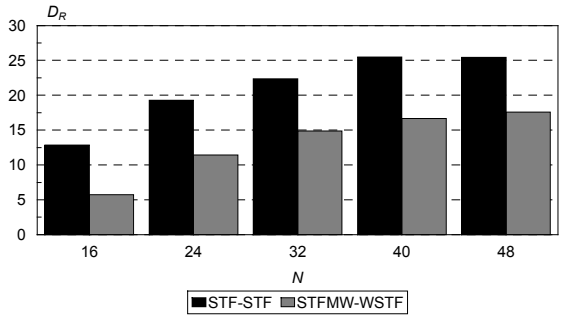

Figure 4. D_R versus N, C=4, Uniform distribution case

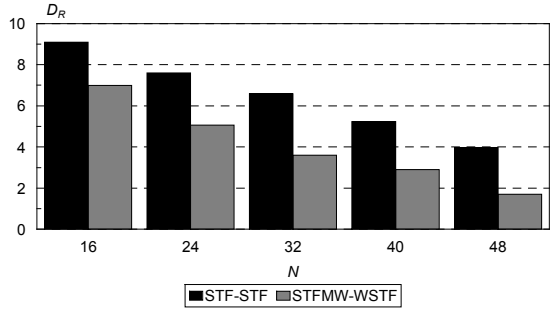

Figure 5. D_R versus N, C=1, Normal distribution case

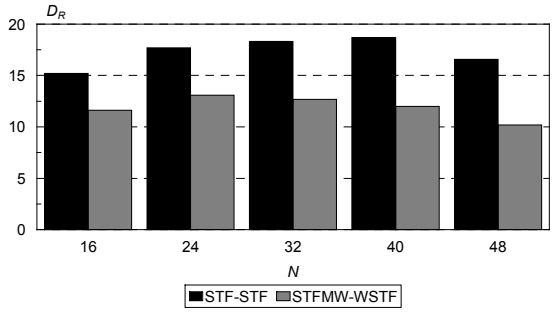

Figure 6. D_R versus N, C=2, Normal distribution case

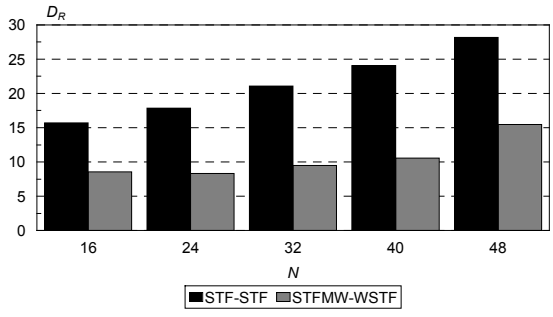

Figure 7. D_R versus N, C=4, Normal distribution case

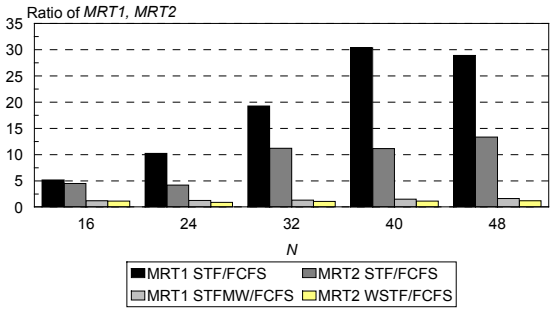

Figure 8. Ratio of *MRT1*, *MRT2* versus *N*, *C*=1, Uniform distribution case

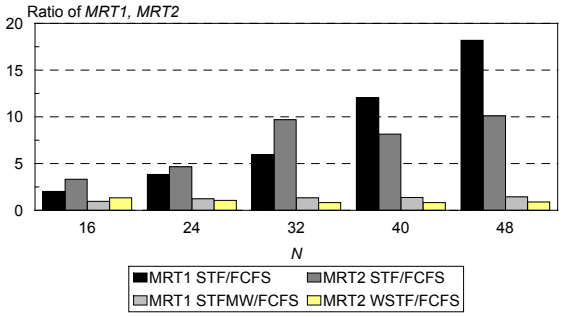

Figure 9. Ratio of *MRT1*, *MRT2* versus *N*, *C*=2, Uniform distribution case

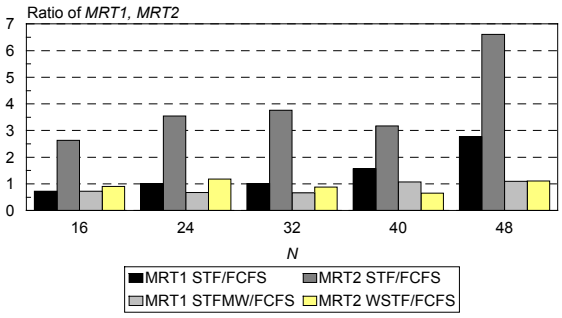

Figure 10. Ratio of *MRT1*, *MRT2* versus *N*, *C*=4, Uniform distribution case

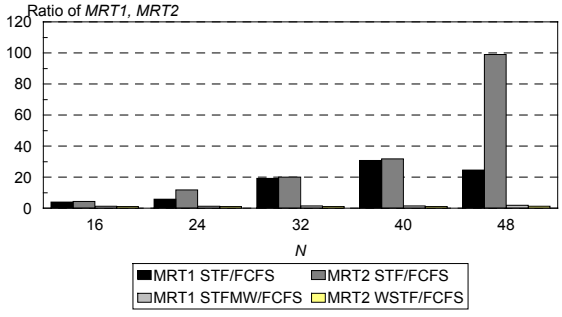

Figure 11. Ratio of *MRT1*, *MRT2* versus *N*, *C*=1, Normal distribution case

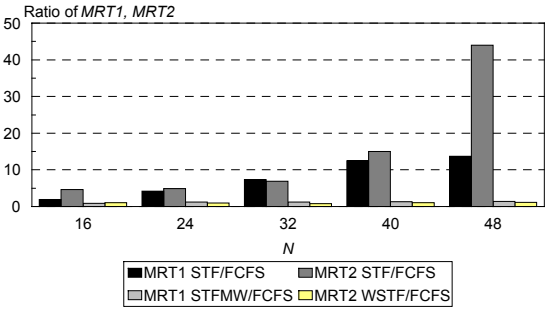

Figure 12. Ratio of *MRT1*, *MRT2* versus *N*, *C*=2, Normal distribution case

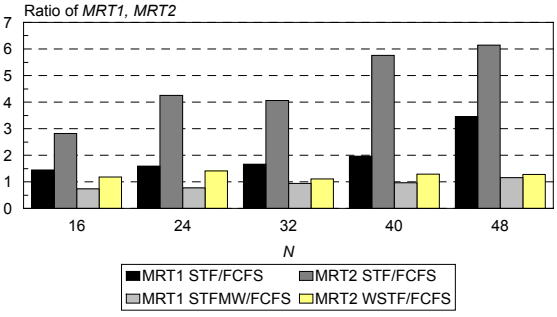

Figure 13. Ratio of *MRT1*, *MRT2* versus *N*, *C*=4, Normal distribution case

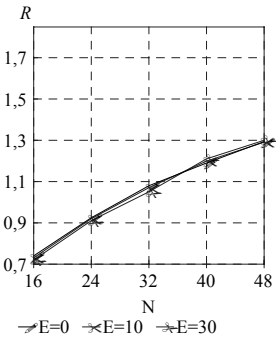

Figure 14. *R* versus *N*, *C*=4, STFMW-WSTF policy, Uniform distribution case

4 CONCLUSIONS AND FURTHER RESEARCH

In this paper, we have investigated the problem of distributed system scheduling. Our objective is to identify conditions that produce good overall system performance while maintaining fairness in individual job execution time. Simulation is used to produce our results. Three policies are considered which combine scheduling at the processors and at the I/O subsystem (FCFS-FCFS, STF-STF, and STFMW-WSTF). Performance for each of these policies was analyzed and compared for

various degrees of multiprogramming, for different coefficients of variation of task execution times, and for two different types of job parallelism. The simulation results indicate that all policies have merit:

- The STF-STF policy improves performance over the FCFS-FCFS method, but does not provide a guarantee of individual job service at the processors and at the I/O unit. In the extreme case, this method delays tasks which are large in comparison to the mean processor service time, and/or where jobs need a large I/O service time. Because processor and I/O queues are rearranged each time a new task (or job) is added, some tasks (or jobs) may never be scheduled. This method is normally used only in production systems where starvation is not an issue.

- STFMW-WSTF does not perform as well as STF-STF, but it performs better than FCFS-FCFS and is fair to jobs. In many cases it yields satisfactory results without causing excessive delays in the system units.

- The relative performance of the three policies depends on the degree of multiprogramming, on the coefficients of variation of task execution times, and on job characteristics that are related to the degree of parallelism.

The analysis presented in this paper is far from complete. Further experimentation is required to examine other cases which would involve preemptive scheduling methods, and I/O service times with higher variability.

REFERENCES

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

Dandamundi S. 1994, "Performance implications of task routing and task scheduling strategies for multiprocessor systems". In *Proc. of the IEEE-Euromicro Conf. on Massively Parallel Computing Systems* (Ischia, Italy, May) IEEE Computer Society, Los Alamitos, CA, USA. Pp348-353.

Karatza H.D. 1997, "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, April, SCSI, San Diego, CA, USA. Pp241-247.

Karatza H.D. 2000a, "Scheduling Strategies for Multitasking in a Distributed System". In *Proc. of the 33rd Annual Simulation Symp.* (Washington D.C., USA, April) IEEE Computer Society, Los Alamitos, CA, USA. Pp 83-90.

Karatza H.D. 2000b, "Gang Scheduling and I/O Scheduling in a Multiprocessor System". In *Proc. of 2000 Symp. on Performance Evaluation of Computer and Telecommunication Systems* (Vancouver, Canada, July) SCSI, San Diego, CA, USA. Pp245-252.

Kumar A. and Shorey R. 1993, "Performance Analysis and Scheduling of Stochastic Fork-Join Jobs in a Multicomputer System", *IEEE Transactions on Parallel and Distributed Systems*, IEEE Computer Society, Los Alamitos, CA, USA, Vol. 4, No. 10. Pp1147-1162.

Kwong P. and Majumdar S. 1999, "Scheduling of I/O in Multiprogrammed Parallel Systems", *Informatica* 23. Pp 67-76.

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

Rosti E., Serazzi G., Smirni E. and Squillante M. 1998, "The Impact of I/O on Program Behavior and Parallel Scheduling". *Performance Evaluation Review*. ACM, New York, USA. Vol. 26 (1). Pp56-65.

Seltzer M., Chen P. and Ousterhout J. 1990, "Disk Scheduling Revisited", In *Proc. of the 1990 Winter Usenix* (Washington D.C., USA, January) Usenix Association, Berkeley, California. USA. Pp313-324.

Sevcik K. 1994, "Application Scheduling and Processor Allocation in Multiprogrammed Parallel Processing Systems", *Performance Evaluation*, Elsevier, Amsterdam, Holland, Vol. 19. Pp107-140.

Worthington B.L., Ganger G.R. and Patt Y.N. 1994, "Scheduling Algorithms for Modern Disk Drives". In *Proc. of the ACM Sigmetrics Conf.* (Nashville, TN, USA, May) ACM, New York, USA. Pp241-251.

BIOGRAPHY

HELEN D. KARATZA is an Assistant Professor in the Department of Informatics at the Aristotle University of Thessaloniki, Greece. Her research interests include Performance Evaluation, Multiprocessor Scheduling and Simulation.