PERFORMANCE ANALYSIS OF GANG SCHEDULING IN A DISTRIBUTED SYSTEM UNDER PROCESSOR FAILURES

HELEN D. KARATZA

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

Abstract: In this paper we study the performance of a distributed system that is subject to hardware failures and subsequent repairs. A special type of scheduling called *gang scheduling* is considered, under which jobs consist of a number of interacting tasks which are scheduled to run simultaneously on distinct processors. Two different gang scheduling policies used to schedule parallel jobs are examined in two cases: In the blocking case, a job that is blocked due to processor failure keeps all of its assigned processors until the failed processor is repaired. In the non-blocking case, the remaining operable processors can serve other jobs. Furthermore, this paper combines two different I/O scheduling methods with the two gang scheduling policies, so that three policy combinations are used. The impact of the variability in processor service time is also studied. Various degrees of multiprogramming, coefficients of variation of processor service time and failure to repair ratios are examined using simulation techniques.

***Keywords*:** Simulation, Performance, Distributed Systems, Gang Scheduling, and Processor Failures.

1 INTRODUCTION

Distributed computing has drawn considerable attention over recent years. However, it is not obvious how to allocate computing nodes of the distributed system among competing jobs. One idea is to use *gang scheduling* where a set of tasks is scheduled to execute simultaneously on a set of processors. It allows tasks to interact efficiently by using busy waiting, without the risk of waiting for a task that is not currently running. Without gang scheduling, tasks have to block in order to synchronize, thus suffering context switch overhead.

The code to simultaneously schedule all tasks of each gang could become overly complex requiring elaborate bookkeeping and global system knowledge. Because gang scheduling demands that no task execute unless all other gang member tasks execute, some processors may remain idle even when there are tasks waiting to be run.

With gang scheduling, at any time there is a one-to-one mapping between tasks and processors. Although the total number of tasks in the system may be larger that the number of processors, no gang contains more tasks than it does processors. We assume that all the tasks within the same gang execute for the same amount of time, i.e., that the computational load is balanced between them. This is clearly different from the task level models [Dandamudi, 1994], and [Karatza, 2000b], in which, after a job arrives to the system, it is immediately split into component tasks. These tasks are processed on any processor in any order as long as precedence constraints are not violated.

A number of gang scheduling policies for distributed systems and multiprogrammed parallel systems have been proposed, each differing in the way resources are shared among the jobs [Feitelson and Rudolph, 1995], [Feitelson and Rudolph, 1996], [Feitelson and Jette, 1997], [Karatza, 1999a], [Karatza, 2000a], [Sobalvarro and Weihl, 1995], [Squillante et al, 1996], and [Wang et al, 1997]. None of these considers processor failures.

Only distributed systems are considered in this paper. Simulation models are used to answer performance questions about systems in which the processors are subject to failure. In environments that are subject to processor failure, any job that has been interrupted by failure must be restart execution. Recovery from failure implies that a newly reactivated processor is reassigned work. When an idle processor fails, it can be immediately removed from the set of available processors and the reassignment of jobs after the processor is repaired is arbitrary.

Many authors such as [Karatza and Hilzer, 1999], [Karatza, 1999b], [Onyuksel and Hosseini, 1995], and [Rosti et al, 1995] also have studied the performance of systems with processor failures. However, distributed systems with gang scheduling that are prone to processor failures have been studied only in [Karatza, 1999b] so far.

We study gang scheduling in a closed queuing network model of a distributed system where I/O equipment is incorporated. The design choices considered include different ways to schedule gangs. We compare the performance of two gang-scheduling policies that are combined with two I/O scheduling strategies for different degrees of multiprogramming, various coefficients of variation of processor service time and different failure to repair ratios. Also, two different ways of employing operable processors when a processor fails are examined. This is an extensive study on this topic. The [Karatza, 1999b] paper does not consider the impact of the variability of job service demand on performance, and it examines lower multiprogramming levels than those in this work. It also considers only one scheduling method at the I/O subsystem. Moreover, in [Karatza, 1999b] the task routing method is based on the shortest queue criteria that needs a considerable amount of overhead to make task assignment decisions that are based on global system state. In this study we employ a different routing method that is probabilistic and therefore much easier to implement in distributed systems of any size.

To our knowledge, the analysis of gang scheduling in distributed environments that incorporate processor failures like these does not appear elsewhere in the research literature.

The structure of this paper is as follows. Section 2.1 introduces system and workload models, section 2.2 describes the scheduling policies and section 2.3 presents the metrics used to assess the performance of the scheduling policies. The model implementation and its input parameters are described in section 3.1, while the simulation results are both presented and analysed in section 3.2. Finally, section 4 summarizes the paper and provides recommendations 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. They are 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 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 behaviors, we expect that various architectural characteristics will be captured.

The degree of multiprogramming is constant during the simulation experiment. A fixed number of jobs N are circulating alternatively between the processors and the I/O unit. Therefore, we examine I/O processing along with parallel job scheduling. This is also proposed by [Rosti et al, 1998]. They study 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. The configuration of the model is shown in Figure 1.

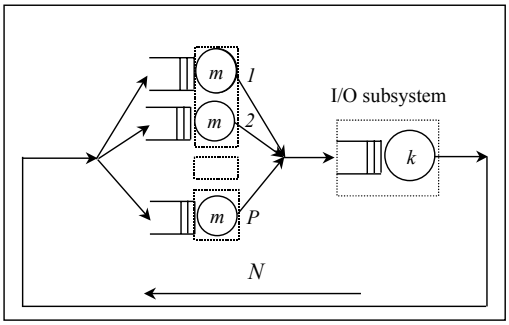

Figure 1: The queuing network model

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

The system is prone to processor failures and processor failure is a Poisson process with a failure rate of α. Processor repair time is an exponentially distributed random variable with the mean value of $1/\beta$. For our model, there are enough

repair stations for all failed processors so that they all can be repaired concurrently.

In this system, simultaneous multiple processor failures are not allowed. Idle and allocated processors are equally likely to fail. If an idle processor fails, it is immediately removed from the set of available processors. It is reassigned only after it has been repaired.

When a processor fails during task execution, all work that was accomplished on all tasks associated with that job needs to be redone. Tasks of failed jobs are resubmitted for execution as the first tasks in the assigned queues.

The number of tasks in a job is the job's *degree of parallelism*. The number of tasks of job x is represented as $t(x)$. If $p(x)$ represents the number of processors required by job x, then the following relationship holds:

$$1 \le t(x) = p(x) \le P$$

The number of tasks in job x is called the "size" of job x. We call a job "small" ("large") if it requires a small (large) number of processors. Each time a job returns from I/O service to the distributed processors, it requires a different number of processors for execution. That is, its degree of parallelism is not constant during its lifetime in the system.

Each task of job x is routed to a different processor for execution. The routing policy is probabilistic and is defined as follows:

Let:

n_assigned = number of tasks of job x that have been already assigned a processor.

Then:

n_assigned = 0;
While n_assigned < t(x) do
Begin
n_processors = P – n_assigned;
Select randomly one of the n_processors;
n_assigned = n_assigned + 1;
i = n_assigned;
Assign the selected processor to the i^{th} task of job x;
Remove this processor from the set of candidate processors for the next task assignment of job x
End.

Tasks in processor queues are examined in order accordingly to the scheduling policy. A job x starts to execute only if all $p(x)$ processors assigned to it are available. Otherwise, all job x tasks wait in their assigned queues. When a job finishes execution, all processors assigned to it are released. The number of jobs that can be processed in parallel depends on the following:

- Job size.
- Number of operable processors.
- Scheduling policy that is employed.

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 gang sizes.
- The distribution of task service demand.
- 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 gang with a small number of tasks may have a long execution time.

2.1.1 Distribution of gang size

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 $\eta = (1+P)/2$.

2.1.2 Service time distribution

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 a proportionately 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 gang with a long service demand starts execution, it occupies its assigned processors for a long time interval, and, depending on the scheduling policy, it may introduce inordinate queuing delays for other tasks waiting for service.

The parameter that represents the variability in task service demand (gang execution time) is the

coefficient of variation of task service demand C. This is the ratio of the standard deviation of gang execution time to its mean.

We examine the following cases with regard to gang execution time distribution:

- Gang execution time is an exponentially distributed random variable with mean m.
- Gang execution time has a Branching Erlang distribution [Bolch, 1998] with two stages. The coefficient of variation is $C > 1$ 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.

All notations used in this paper appear in Table 1.

2.2 Scheduling Strategies

We assume that the scheduler has perfect information when making decisions, i.e. it knows the exact number of processors required by each job and also the service time of I/O service requests.
We now describe the scheduling strategies employed. We assume that the scheduling overhead is negligible.

For processor scheduling, the following gang scheduling policies are considered:

***Adapted First-Come-First-Served* (*AFCFS*)**. This method attempts to schedule a job whenever processors assigned to its tasks are available. When there are not enough processors available for a large job whose tasks are waiting in the front of the queues, AFCFS policy schedules smaller jobs whose tasks are behind the tasks of the large job. One major problem with this scheduling policy is that it tends to favor those jobs requesting a smaller number of processors and thus may increase fragmentation of the system.

***Largest-Gang-First-Served* (*LGFS*)**. With this policy tasks are placed in increasing job size order in processor queues (tasks that belong to larger gangs are placed at the head of queues). All tasks in queues are searched in order, and the first jobs whose assigned processors are available begin execution. This method tends to improve the performance of large, highly parallel jobs at the expense of smaller jobs, but in many computing environments this discrimination is acceptable, if not desirable. For example, supercomputer centers often run large, highly parallel jobs that cannot run elsewhere.

When a processor fails during task execution, all tasks of the corresponding job are resubmitted for execution as the leading tasks in the assigned queues. They wait at the head of these ready queues until the failed processor is repaired. During that time there are two cases for each of the AFCFS and LGFS policies:

Blocking case. The remaining processors assigned to the interrupted job are blocked and cannot execute other job tasks. Unfortunately, this case is conservative since jobs are only retained on processor queues when they could run on those processors.

Non-blocking case. Jobs in queues are processed early instead of requiring them to wait until the blocked job resumes execution. The remaining processors assigned to the blocked job execute tasks of other jobs waiting in their queues. This case incurs additional overhead since it can examine all jobs in these queues when a processor fails.

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

***First-Come-First-Served* (*FCFS*)**. This disk scheduling policy often results in sub-optimal performance. However it is fair to jobs.

***Shortest-Time-First* (*STF*)**. This method 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 assumes a-priori knowledge about an I/O request in form of service time. When such knowledge is available, jobs in the I/O queue are ordered in a decreasing order of service time. However, it should be noted that a-priori information is often not available and only an approximation of I/O service time is available. I/O estimated service times are uniformly distributed within $\pm E\%$ of the exact value.

The following policy combinations are used (for processor and I/O subsystem scheduling policies respectively):

- ***AFCFS-FCFS***. This is the fairest of all methods that we examine.

- ***LGFS-FCFS***. This method is not fair to jobs in the processor queues but it provides fairness at the I/O subsystem.

- ***LGFS-STF***. This policy is not fair in terms of both processor and I/O service, but it can be applied in production systems.

When one of the above policies (for example AFCFS-FCFS) is employed with the Blocking Case, then a B in parenthesis (AFCFS-FCFS(B)) follows the policy notation.

2.3 Performance Metrics

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 this job (time spent in processor queues plus time spent in service).
- *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

K	Mean cycle time
R	System throughput
U	Mean processor utilisation
N	Degree of multiprogramming
a	Failure rate
$1/\beta$	Mean repair time
P	Number of processors
m	Mean processor service time
k	Mean I/O service time

Overall system performance is determined by system throughput. The mean cycle time represents program performance. Internal efficiency is primarily influenced by processor utilization because it indicates the level of contention for critical system resources. When each of the LGFS-FCFS and LGFS-STF policies is compared to the AFCFS-FCFS(B) case, 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 modeling [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 1.88235 jobs are served each unit of time. This implies that I/O mean service time must be equal to 1/1.88235 = 0.531 if the I/O unit is to have the same service capacity.

The system is 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 I/O service time is required, we have also examined estimation errors of ±10%, ±20%, and ±30%.

In typical systems, processor failures and repairs do not occur very frequently. In order to produce a sufficient number of data points for these rare events, the simulation program was run for 20,000,000 job services at the processors. A value of $\alpha = 10^{-3}$ is used (i.e., mean inter-failure time or $1/\alpha = 10^3$). The failure to repair ratio (or $\alpha/\beta = \varphi$) is set at 0.05, and 0.1, which means mean repair times (or $1/\beta$) are set to 50, and 100.

3.2 Performance Analysis

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

- Table 2 shows the mean processor utilization range for all N in the $\varphi = 0.10$ case, for C = 1 and C = 4.

- In Figures 2-7, R versus N is depicted in the $\varphi = 0.10$ case.
- Figures 8-10 show the ratio of K in each one of the LGFS-FCFS and LGFS-STF cases over the corresponding value in the AFCFS-FCFS(B) case when $\varphi = 0.10$.
- Figures 11-16 represent D_R versus N in the $\varphi = 0.05$ and $\varphi = 0.10$ cases.

The following conclusions are derived from the results:

In all cases, system performance is better in the $\varphi = 0.05$ case than in the case of $\varphi = 0.10$. This is because when a processor fails, not only tasks of this processor queue are delayed, but also tasks in other processor queues that have a sibling task waiting for service in the failed processor queue. The larger the repair time, the higher the probability is that some gangs will start execution if that processor is available to one of their tasks.

The difference in performance between each one of AFCFS-FCFS, LGFS-FCFS, and LGFS-STF and its corresponding blocking policy is not significant. The relative difference in performance varies within a 0.5-1.6 (%) range. Reasons for this are:

First, when a processor fails, the number of operable processors decreases by one. Therefore, every job that returns to processors after I/O service and needs all of the processors to execute, cannot be served during the time that one processor is down. Since the probability for any job to have gang size equal to P is equal to $1/P$, during the repair time of the failed processor some jobs in one of their visits to processors may have gang size equal to P. These jobs have to wait in processor queues until the failed processor recovers, no matter if a blocking or a non-blocking scheduling policy is employed.

Also, the interrupted job in the blocking case resumes execution as soon as the failed processor recovers. However, in the non-blocking case, that job may have an elongated response time. This is because when the failed processor recovers, some of the processors assigned to that job may have already worked on other jobs. Those jobs will not finish at the same time and therefore, their assigned processors are not used efficiently.

The mean processor utilization is slightly higher in the non-blocking case than in the blocking case (Table 2). However, in both cases, part of the processor utilization is repeat work caused by processor failures rather than useful work.

In both the blocking and non-blocking cases, the LGFS-FCFS method performs better than AFCFS-FCFS. Therefore, the gang scheduling method LGFS performs better than the AFCFS. This is because the mean response time of jobs is lower in the LGFS policy case than when AFCFS is employed. This produces a lower mean cycle time and therefore better overall performance when the LGFS policy is used. The LGFS-STF method performs better than the LGFS-FCFS. This is due to the fact that the STF I/O scheduling strategy performs better than the FCFS. STF yields lower mean delay of jobs at the I/O subsystem than FCFS, which results in lower mean cycle times. I/O scheduling definitely affects performance.

In all cases, it is evident that the AFCFS-FCFS(B) strategy yields the worst performance while the best performance is provided by the LGFS-STF method. Furthermore, the second best method after LGFS-STF is LGFS-FCFS. The relative difference in performance of these policies is depicted in Figures 11-16 (with regard to system throughput), and also in Figures 8-10 (with regard to mean cycle time).

The I/O scheduling benefits are better exploited when gangs have high variability in their service time. This is due to the fact that when a gang with a very large service demand is executed on some processors, it may introduce inordinate queuing delays to other gangs, which have very small service demands. During this time the I/O subsystem may starve but later it can become deluged with jobs that spend a large amount of time waiting in the I/O queue. In this case, the STF I/O scheduling mechanism alleviates the problem better than the FCFS I/O scheduling policy.

Additional simulation experiments were conducted to assess the impact of I/O service time estimation error on the performance of the LGFS-STF method. The estimation error was set at ±0%, ±10%, ±20%, and ±30%. The results lead us to the same conclusions reached in previous research [Karatza, 2000a], that the estimation error in I/O service time only marginally affects system performance.

When a blocking policy is compared with a non-blocking policy, it is necessary to take into account that the non-blocking policy incurs additional overhead as all jobs in the queues are examined when a processor fails. This overhead is not modeled in this paper because processor failure is a rare event and therefore, it is not expected

to seriously affect the performance of a non-blocking policy. However, when we also take into account that non-blocking gang scheduling performs only marginally better than blocking scheduling, the blocking case is superior since it is easier to be implemented.

Table 2: Mean processor utilization range

Scheduling Policy	*U* range $\varphi = 0.10$	
	$C = 1$	$C = 4$
AFCFS-FCFS(B)	0.629-0.670	0.536-0.600
LGFS-FCFS(B)	0.644-0.678	0.539-0.613
LGFS-STF(B)	0.647-0.679	0.569-0.629
AFCFS-FCFS	0.632-0.678	0.539-0.606
LGFS-FCFS	0.647-0.688	0.542-0.621
LGFS-STF	0.651-0.689	0.572-0.637

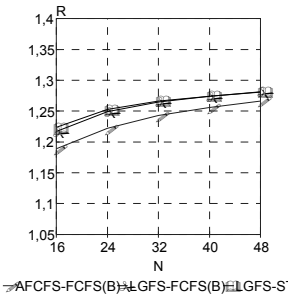

Figure 2: *R* versus *N*, *C*=1, $\varphi = 0.10$, Blocking case

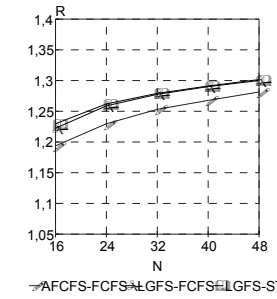

Figure 3: *R* versus *N*, *C*=1, $\varphi = 0.10$, Non-Blocking case

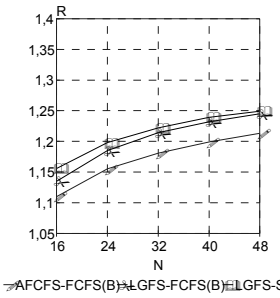

Figure 4: *R* versus *N*, *C*=2, $\varphi = 0.10$, Blocking case

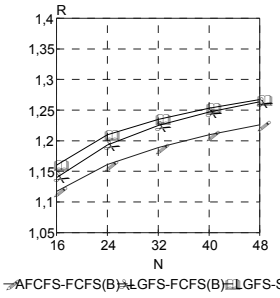

Figure 5: *R* versus *N*, *C*=2, $\varphi = 0.10$, Non-Blocking case

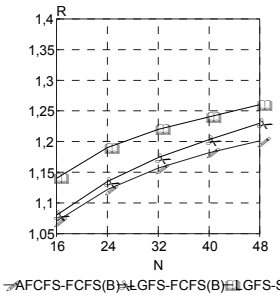

Figure 6: *R* versus *N*, *C*=4, $\varphi = 0.10$, Blocking case

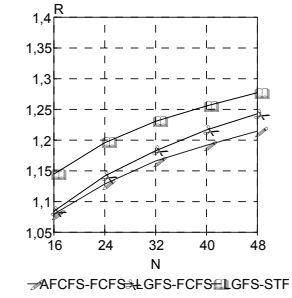

Figure 7: *R* versus *N*, *C*=4, $\varphi = 0.10$, Non Blocking case

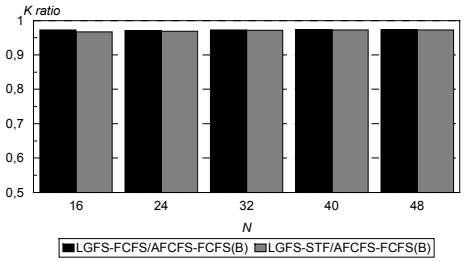

Figure 8: Ratio of K versus N, C=1, φ = 0.10

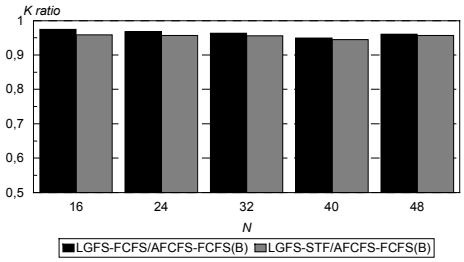

Figure 9: Ratio of K versus N, C=2, φ = 0.10

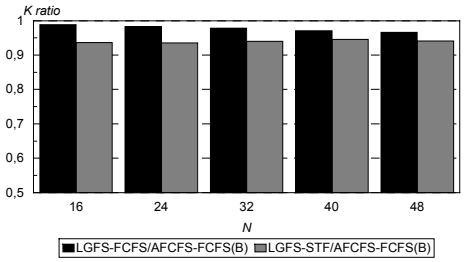

Figure 10: Ratio of K versus N, C=4, φ = 0.10

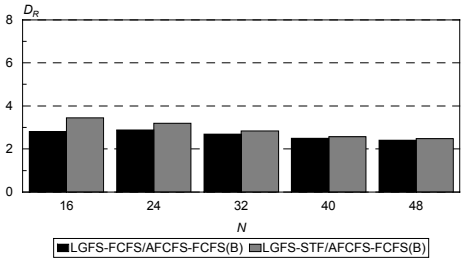

Figure 11: D_R versus N, C=1, φ = 0.05

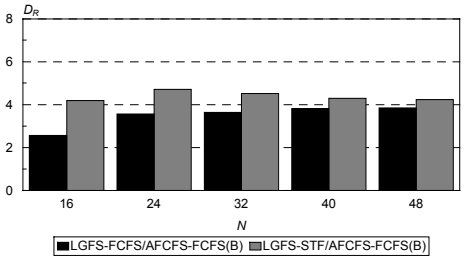

Figure 12: D_R versus N, C=2, φ = 0.05

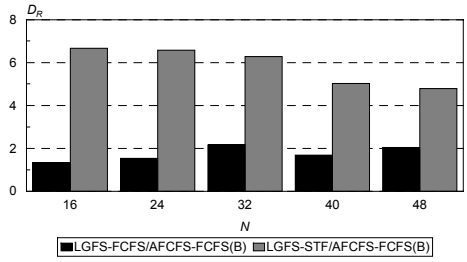

Figure 13: D_R versus N, C=4, φ = 0.05

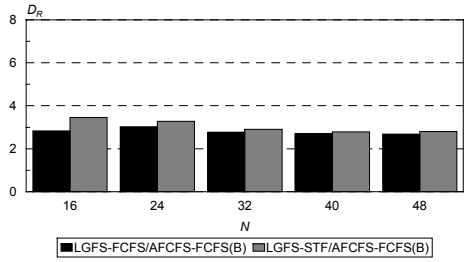

Figure 14: D_R versus N, C=1, φ = 0.10

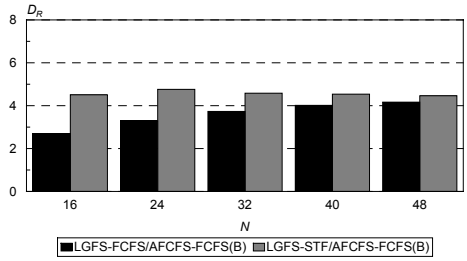

Figure 15: D_R versus N, C=2, φ = 0.10

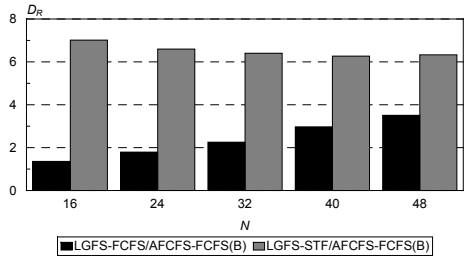

Figure 16: D_R versus N, C=4, φ = 0.10

4 CONCLUSIONS AND FURTHER RESEARCH

This paper examines the performance of three policies, which combine gang scheduling and I/O scheduling in a distributed system that encounters processor failures. The following two processor failure cases are examined: The block-

ing case where a job that is stopped due to a processor failure keeps all its assigned processors until the failed processor is repaired, and the non-blocking case where the remaining operable processors can serve other jobs. Various degrees of multiprogramming, coefficients of variation of processor service time and failure to repair ratios are examined using simulation techniques.

The following is a summary of the simulation results:

- In all cases, the Largest-Gang-First-Served method combined with the Shortest-Task-First I/O scheduling policy outperforms the other methods.

- With respect to blocking/non-blocking the service of jobs when a processor fails, the blocking case is preferred as it is easier to implement and performs close to the level of the non-blocking case.

- The impact of I/O scheduling seems to be more significant when the variability in gang service time is high.

This research could be extended to include cases where after a failed processor recovers, the blocked job immediately resumes execution by pre-empting all jobs using its assigned processors. It could also consider different distributions of I/O service times.

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.

Feitelson D.G. Rudolph R. 1995, "Parallel Job Scheduling: Issues and Approaches". In *Job Scheduling Strategies for Parallel Processing*, *Lecture Notes in Computer Science*, Springer-Verlang, Berlin, Germany. Vol. 949. Pp1-18.

Feitelson D.G. and Rudolph L. 1996. "Evaluation of Design Choices for Gang Scheduling Using Distributed Hierarchical Control". *Journal of Parallel and Distributed Computing*, Academic Press, New York, USA, Vol. 35. Pp18-34.

Feitelson D.G. and Jette M.A. 1997, "Improved Utilisation and Responsiveness with Gang Scheduling". In *Job Scheduling Strategies for Parallel Processing*, *Lecture Notes in Computer Science*, Springer-Verlang, Berlin, Germany. Vol. 1291. Pp238-26.

Karatza H.D. and Hilzer. R.C. 1999. "Processor Failures in a Shared - Memory Multiprocessor System with Resequencing". In *Proc. of the High Performance Computing Symp. '99* (San Diego, CA, USA, April) SCSI, San Diego, CA, USA. Pp243-248.

Karatza H.D. 1999a, "A Simulation-Based Performance Analysis of Gang Scheduling in a Distributed System". In *Proc. of the 32nd Annual Simulation Symp.* (San Diego, CA, USA, April) IEEE Computer Society, Los Alamitos, CA, USA. Pp26-33.

Karatza, H.D. 1999b, "Gang Scheduling in a Distributed System with Processor Failures". In *Proc. of the UK Performance Engineering Workshop* (Bristol, UK, July) University of Bristol, Bristol, UK. Pp199-208.

Karatza H.D. 2000a, "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.

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

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

Onyuksel I. and Hosseini S.H. 1995. "A Simulation Model for Performance Evaluation of Fault-Tolerant Computer Systems". In *Proc. of the Summer Computer Simulation Conf.* (Ottawa, Ontario, Canada, July) SCSI, San Diego, CA, USA. Pp795-800.

Rosti E., Smirni E., Serazzi G. and Dowdy L.W. 1995, "Analysis of Non-Work-Conserving Processor Partitioning Policies". In *Proc. of the Workshop on Job Scheduling Strategies for Parallel*

Processing (Santa Barbara, CA, USA, April) IEEE Computer Society, Los Alamitos, CA, USA. Pp165-181.

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.

Sobalvarro P.G. and Weihl W.E. 1995, "Demand-based Coscheduling of Parallel Jobs on Multiprogrammed Multiprocessors". In *Job Scheduling Strategies for Parallel Processing*, *Lecture Notes in Computer Science*, Springer-Verlang, Berlin, Germany. Vol. 949. Pp106-126.

Squillante M.S., Wang F. and Papaefthymioy M. 1996, "Stochastic Analysis of Gang Scheduling in Parallel and Distributed Systems", *Performance Evaluation*, Elsevier, Amsterdam, Holland, Vol. 27&28 (4). Pp273-296.

Wang F., Papaefthymiou M. and Squillante M.S. 1997, "Performance Evaluation of Gang Scheduling for Parallel and Distributed Systems". In *Job Scheduling for Parallel Processing*, *Lecture Notes in Computer Science*, Springer-Verlang, Berlin, Germany. Vol. 1291. Pp184-195.

BIOGRAPHY

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