Gang Scheduling in a Distributed System under Processor Failures and Time-varying Gang Size

Helen D. Karatza
Department of Informatics
Aristotle University of Thessaloniki
54124 Thessaloniki, Greece
karatza@csd.auth.gr

Abstract

In this paper we study the performance of a distributed system which 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. System performance is examined and compared in cases where different distributions for the number of parallel tasks per job (gang size) are employed. We examine cases where gang size is defined by a specific distribution and also a case where gang size distribution varies with time.

1. Introduction

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 jobs. Good scheduling policies are needed to improve system and program performance. The scheduling of parallel jobs on distributed processors has always been an important and challenging area of research. In this study jobs consist of parallel tasks that are scheduled to execute concurrently on a set of processors. The parallel tasks need to start at essentially the same time, co-ordinate their execution, and compute at the same pace. This type of resource management is called "gang scheduling" or "co-scheduling" and has been extensively studied in the literature.

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 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 can be arbitrary.

The main purpose of this paper is to study and compare the performance of different scheduling algorithms under different cases of job parallelism. We pursue to investigate whether the method that performs better than the others for one type of workload, is also the best method for the other types of workload that we examine. Three cases of job parallelism are considered: in one case, jobs have highly variable degree of parallelism during their lifetime, while in the second case, the majority of jobs exhibit a moderate parallelism. In the third case, job parallelism is not defined by a specific pattern but changes with the time. So, a time interval during which arriving jobs exhibit highly variable degrees of parallelism, is followed by a time interval during which the majority of jobs exhibit moderate parallelism, and vice-versa.

In previous works we studied gang scheduling methods in distributed systems where gang parallelism was highly variable. For this purpose we used the uniform distribution for the number of tasks per job. However, in real systems, the variability in job parallelism maybe low, or it can vary during the day depending on the jobs that run on different time intervals. For this reason in this paper we also employ the normal distribution and an exponentially varying with the time distribution which represents real parallel system workloads. We are particularly interested for the last case of job parallelism. Also, the performance of different scheduling policies is compared for various coefficients of variation of the processor service times and for different degrees of multiprogramming.

Gang scheduling is studied extensively in [1-3], and [7-9]. However, these works do not consider processor failures. [4], and [5] study gang scheduling under processor failures, but they consider only the uniform distribution for the gang size. Furthermore, [4] and [5] study smaller degrees of multiprogramming than those examined here. The routing policy that is employed in [4] is based on the shortest queue criterion, while the routing policy that is employed in [5] is based on probabilities. Time-varying distribution for the gang size is considered in [6], but that paper does not consider processor failures. Furthermore the system that is studied in [6] is a shared memory partitionable parallel system where all jobs share a single queue, whereas this paper considers a distributed system because each processor is equipped with its own queue. To our knowledge, the analysis of gang scheduling in distributed systems under time-varying workload and processor failures does not appear elsewhere in the research literature.

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 that are studied. Model implementation and input parameters are described in section 3 while the results of the simulation experiments are presented and analyzed in section 4. Section 5 is the conclusion and provides suggestions for further research, and the last section is references.

2. Model and methodology

2.1. System and workload models

A closed queuing network model of a distributed system is considered (Figure 1). The model represents a distributed system because each processor has its own queue (memory). There are P = 16 homogeneous and independent processors. 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. 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.

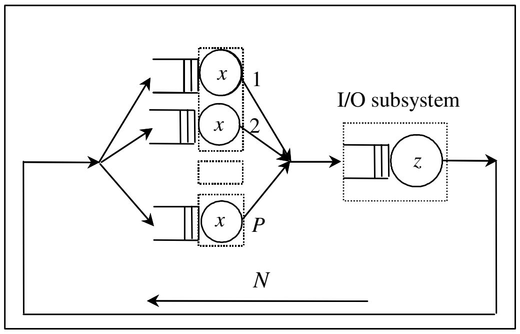

Figure 1. The queuing network model

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 in a job is called the "size" of the job. 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 a job is routed to a different processor for execution. The routing policy is that tasks enter the shortest queues. Tasks in processor queues are examined in order accordingly to the scheduling policy. A job starts to execute only if all processors assigned to it are available. Otherwise, all tasks of the job wait in the assigned queues. When a job finishes execution, all processors assigned to it are released.

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 the following parameters:

- The distribution of gang sizes.
- The distribution of task service demand.
- The distribution of I/O service time.
- The distribution of processor failures.
- The distribution of processor repair time.
- The degree of multiprogramming.

2.1.1. Distribution of gang size. Three different types of distribution of gang size have been utilized:

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

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 $\sigma = \eta/4$.

Exponentially varying distribution. We assume that the distribution of the number of tasks per job changes in exponentially distributed time intervals, from uniform to normal and vice-versa. Those jobs that arrive at the processors within the same time interval have the same distribution of gang size. However, during the same time interval there may exist some jobs at the processors that arrived during a past time interval and which may have a different distribution of gang size. These jobs may still wait at the processors queues or are being served. The mean time interval for distribution change is *d*. Figure 2 shows the changing of gang size distribution with time *t*.

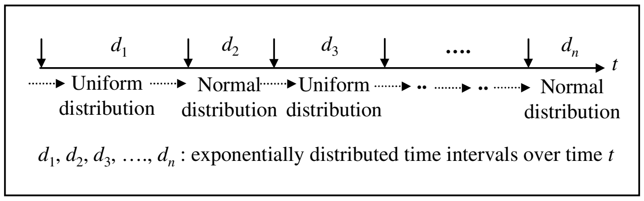

Figure 2. Exponentially time-varying distribution of gang size

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). On the other hand, in the case where the distribution of gang size changes during time, for some exponentially distributed time intervals jobs have highly variable degree of parallelism, while during other intervals the majority of jobs have a moderate parallelism as compared with the number of processors.

2.1.2. Distribution of task service demand. We examine the impact of the variability in task service demand (gang execution time) on system performance. A high variability in task service demand implies that there are 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 is the coefficient of variation of task service demand *C*. We examine the following cases with regard to task service demand distribution:

- Task service demand is an exponentially distributed random variable with mean *x*.
- Task service demand has a Branching Erlang distribution with two stages. The coefficient of variation is *C* >1 and the mean is *x*.

2.1.3. Distribution of I/O service time. After a job leaves the processors, it requests service on the I/O unit. The I/O service times are exponentially distributed with mean *z*.

2.1.4. The distribution of processor failures. Processor failure is a Poisson process with a failure rate of α.

2.1.5. The distribution of processor repair time. Processor repair time is an exponentially distributed random variable with the mean value of $1/\beta$.

2.2. Scheduling strategies

We present two well known queuing policies when gang scheduling is used. A queuing policy is a set of rules that prioritizes the order with which jobs are selected for execution. We assume that the scheduler knows the exact number of processors required by all jobs in the queues.

Adaptive 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.

In order to distinguish the scheduling policies in the two different cases we use the notations AFCFS(B) and LGFS(B) for the blocking case, while we use the notations AFCFS and LGFS in the non-blocking cases.

For the I/O subsystem, the FCFS policy is employed.

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 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 utilization
N	Degree of multiprogramming
α	Failure rate
$1/\beta$	Mean repair time
φ	The failure to repair ratio (α/β)
x	Mean processor service time
z	Mean I/O service time
d	Mean time interval for distribution change
P	Number of processors
D_R	The relative (%) increase in R when policy Y performs better than policy X
D_K	The relative (%) decrease in K when policy Y performs better than policy X

Overall system performance is determined by R. K represents program performance.

3. Experimental methodology

The queuing network model is simulated with discrete event simulation modelling using the independent replication method. The system considered is balanced (refer to Table 1 for notations): $x = 1.0$, $z = 0.531$. The reason $z = 0.531$ 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, .., 80. 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 φ) is set at 0.05, and 0.1, which means mean repair times (or $1/\beta$) are set to 50, and 100.

In the varying distribution case, the mean time interval for distribution change is considered to be d = 10, 20, 30. These are reasonable choices considering that the mean service time of tasks in equal to one.

4. Performance analysis

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

Tables 2 and 3 show the mean processor utilization range for all N in the $\varphi = 0.10$ case, for $C = 1$ and $C = 4$ respectively (see Table 1 for notations).

The relative increase in R between the non-blocking case and the blocking case of each method for the time-varying distribution case is shown in Figures 3, and 4, for $\varphi = 0.10$, $d = 30$, and $C = 1, 4$ respectively. The relative increase in R when each of LGFS(B), AFCFS, and LGFS is employed instead of AFCFS(B) is depicted in Figures 5, 6, and 7, (9, 10, and 11) in the $\varphi = 0.10$, $d = 30$, and $C = 1$, (and $C = 4$) cases respectively. Figures 8, and 12 show the relative decrease in K in the time-varying distribution case when each of LGFS(B), AFCFS, and LGFS is employed instead of AFCFS(B) for $\varphi = 0.10$, $d = 30$, and $C = 1$, and $C = 4$ respectively. Figures 13, 14, and 15 represent D_R versus N, in the $\varphi = 0.10$, $C = 2$, and d = 10, 20, and 30 cases respectively. Figure 16 shows D_R versus N in the $\varphi = 0.05$, $C = 2$, $d = 20$ case.

The following conclusions are drawn from the results.

With each scheduling method, the mean processor utilization is slightly higher in the non-blocking case than in the blocking case (Tables 2, and 3). However, in both cases part of the processor utilization is repeat work caused by processor failures rather than useful work.

In Figures 3 and 4 it is shown that in the varying distribution case for $\varphi = 0.10$, the difference in performance between each one of AFCFS, and LGFS and the corresponding blocking case is less than 6.5%. The smallest difference in performance is observed at $N = 16$ and is less than 4%. In the uniform (normal) distribution case for the same N the difference is less than 3 (less than 5.5) respectively. The reasons that blocking does not significantly degrade performance are the following: Every job which has a task assigned to the failed processor has to wait until the failed processor recovers, no matter if a blocking or a non-blocking scheduling policy is employed. 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 work on other jobs already. Those jobs will not finish at the same time and therefore, their assigned processors will not be used efficiently.

In all cases, the LGFS method performs better than the other methods, while the worst performance is encountered with the AFCFS(B) policy. This is because the mean response time of jobs is lower (higher) in the LGFS (AFCFS(B)) policy case than in the other methods cases. This results in a lower (higher) mean cycle time respectively, and therefore in better (worst) overall performance.

The relative performance of LGFS(B) and AFCFS depends on the workload. In some cases the second best method after LGFS is LGFS(B) while in other cases AFCFS is the second best method. For example, Figures 5-7, and 9-11 show that: In the uniform distribution case, for $C = 1$ LGFS(B) outperforms AFCFS, while for $C = 4$, AFCFS outperforms LGFS(B). In the normal distribution case, for $C = 1$ and for small N AFCFS outperforms LGFS(B) but as N increases the superiority of AFCFS over LGFS(B) is decreasing and LGFS(B) tends to outperform AFCFS. For $C = 4$ in the same distribution case AFCFS outperforms LGFS(B) but the difference in performance is decreasing with increasing N. In the time-varying distribution case AFCFS outperforms LGFS(B), but the difference in performance decreases as N increases. This relative difference in performance is smaller in the $C = 1$ case than in the $C = 4$ case.

With regard to mean cycle time, the relative difference in performance of the scheduling policies for the time-varying distribution case is depicted in Figures 8 and 12. The conclusions are similar to those derived from the values of D_R.

Generally, the superiority of LGFS(B), LGFS, and AFCFS over AFCFS(B) is increasing with increasing N. This is due to the fact that the advantages of these policies as compared to AFCFS(B) are better exploited in the cases of larger queues than in the cases of smaller queues.

In the results presented in Figures 13, 14, and 15 for the time-varying distribution case, we observe that the relative performance of the different policies is not significantly affected by the size of the mean time interval for distribution change. Also, simulation results presented in Figures 14 and 16 show that the relative performance of LGFS(B) and AFCFS depends on φ.

When we compare a blocking policy with the respective non-blocking policy we should take into account that the non-blocking policy incurs an additional overhead as all jobs in the queues may be examined when a processor fails. This overhead has not been modelled in this paper. Therefore, in the cases where non-blocking gang scheduling does not perform significantly better than blocking scheduling, the blocking case should be preferred since it is easier to be implemented.

Table 2. *U* range, *C* = 1, *φ* = 0.10, *d* = 30

	Uniform	Normal	Time-varying
AFCFS(B)	0.623-0.678	0.543-0.571	0.586-0.629
LGFS(B)	0.645-0.721	0.552-0.604	0.600-0.664
AFCFS	0.638-0.716	0.571-0.603	0.607-0.665
LGFS	0.663-0.768	0.581-0.639	0.623-0.705

Table 3. *U* range, *C* = 4, *φ* = 0.10, *d* = 30

	Uniform	Normal	Time-varying
AFCFS(B)	0.562-0.640	0.530-0.562	0.548-0.606
LGFS(B)	0.567-0.669	0.531-0.581	0.551-0.628
AFCFS	0.575-0.673	0.554-0.591	0.567-0.639
LGFS	0.579-0.706	0.556-0.613	0.570-0.663

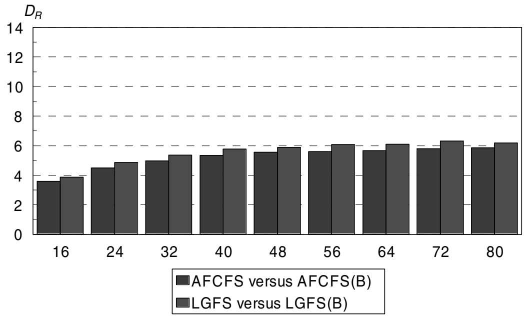

Figure 3. D_R versus N, d = 30, C = 1, φ = 0.10, Time-varying distribution

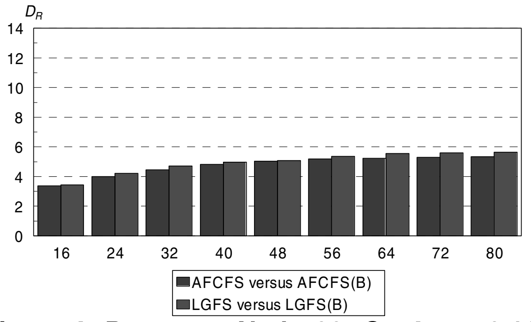

Figure 4. D_R versus N, d = 30, C = 4, φ = 0.10, Time-varying distribution

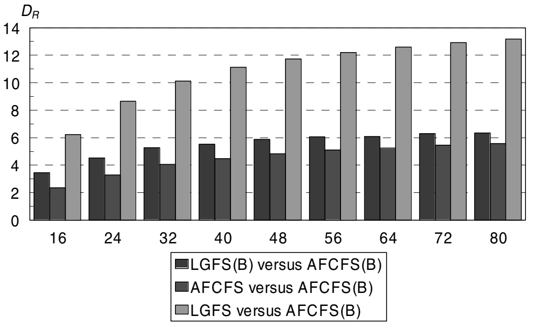

Figure 5. D_R versus N, C = 1, φ = 0.10, Uniform distribution

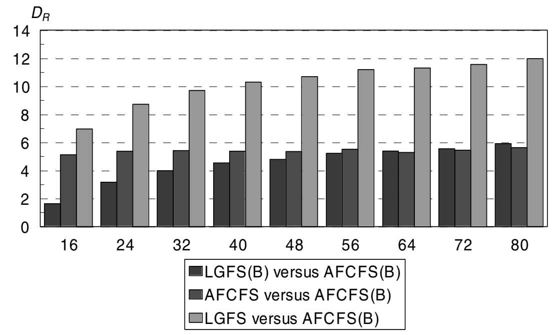

Figure 6. D_R versus N, C = 1, φ = 0.10, Normal distribution

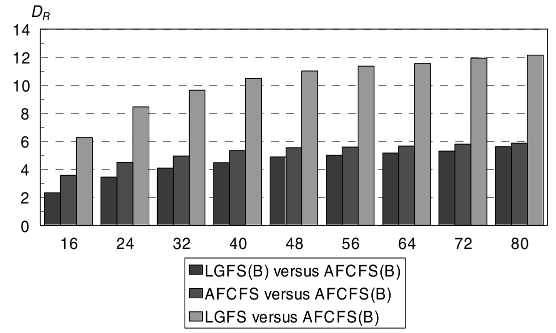

Figure 7. D_R versus N, d = 30, C = 1, φ = 0.10, Time-varying distribution

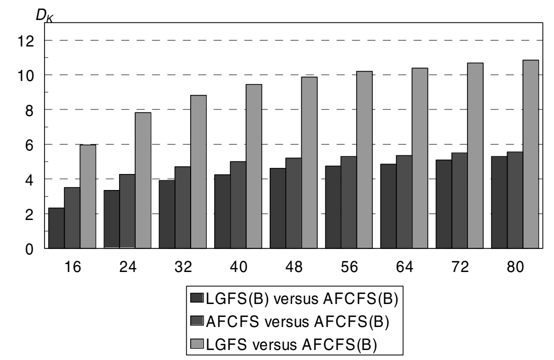

Figure 8. D_K versus N, d = 30, C = 1, φ = 0.10, Time-varying distribution

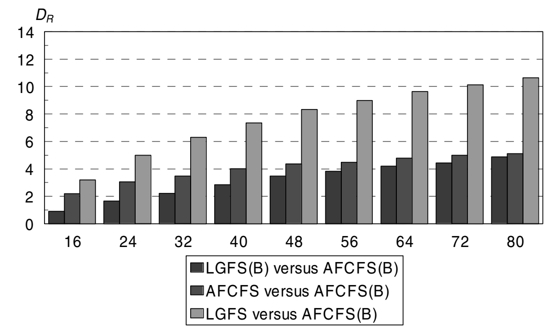

Figure 9. D_R versus N, C = 4, φ = 0.10, Uniform distribution

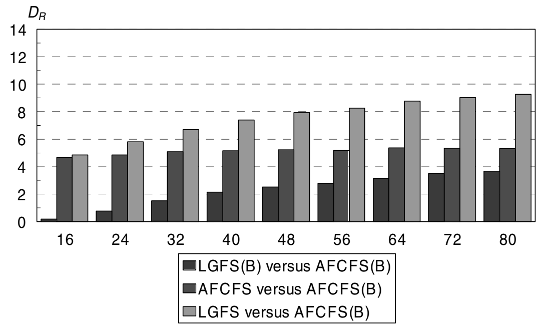

Figure 10. D_R versus N, C = 4, φ = 0.10, Normal distribution

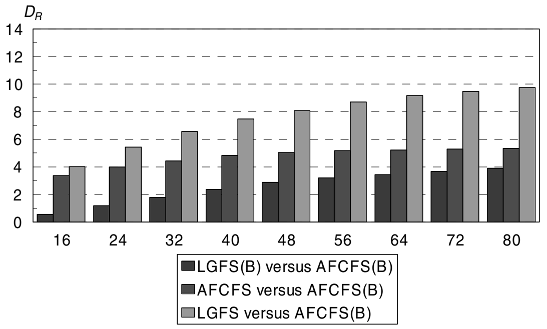

Figure 11. D_R versus N, d = 30, C = 4, φ = 0.10, Time-varying distribution

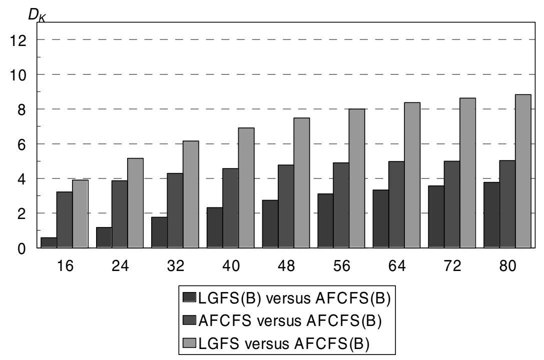

Figure 12. D_K versus N, d = 30, C = 4, φ = 0.10, Time-varying distribution

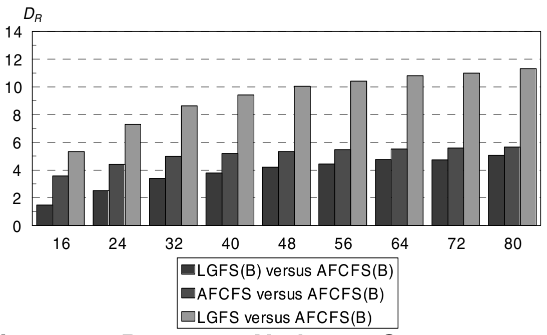

Figure 13. D_R versus N, d = 10, C = 2, φ = 0.10, Time-varying distribution

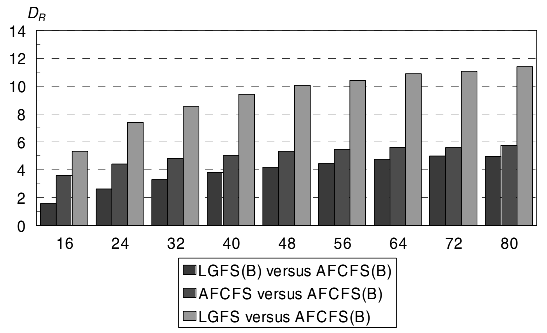

Figure 14. D_R versus N, d = 20, C = 2, φ = 0.10, Time-varying distribution

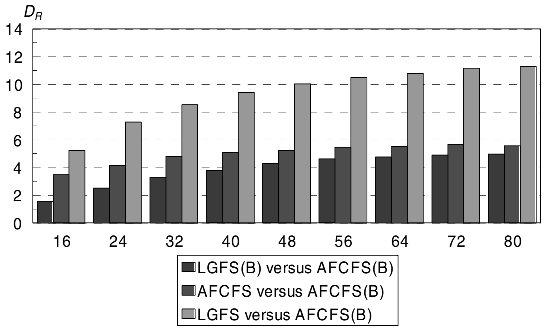

Figure 15. D_R versus N, d = 30, C = 2, φ = 0.10, Time-varying distribution

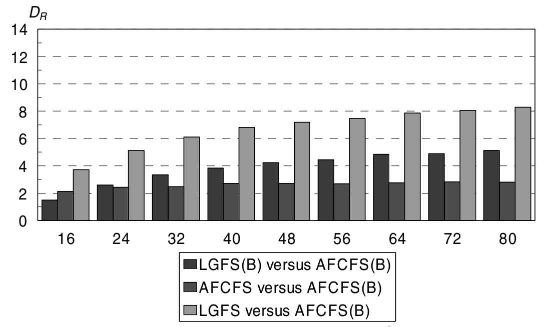

Figure 16. D_R versus N, d = 20, C = 2, φ = 0.05, Time-varying distribution

5. Conclusions and further research

The simulation results reveal that in all cases of gang size distribution that we examine, time-varying, uniform, and normal, the LGFS method outperforms the other methods. Furthermore, the simulation results show that the relative performance of the scheduling policies depends on the workload. We also conclude that in the cases where non-blocking gang scheduling does not significantly outperform blocking scheduling, the blocking case should be preferred as it does not incur the additional overhead associated with the non-blocking gang scheduling.

In this paper we consider a time-varying distribution only for gang size. A logical extension is to also examine the case of a time-varying distribution for task service demand.

6. References

[1] D.G. Feitelson, and R. Rudolph, “Parallel Job Scheduling: Issues and Approaches”, In *Job Scheduling Strategies for Parallel Processing*, *Lecture Notes in Computer Science*, Springer-Verlang, Berlin, Germany, 1995, Vol. 949, pp. 1-18.

[2] D.G. Feitelson, and L. Rudolph, “Evaluation of Design Choices for Gang Scheduling Using Distributed Hierarchical Control”, *Journal of Parallel and Distributed Computing*, Academic Press, New York, USA, 1996, Vol. 35, pp. 18-34.

[3] D.G. Feitelson, and M.A. Jette, “Improved Utilisation and Responsiveness with Gang Scheduling”, In *Job Scheduling Strategies for Parallel Processing*, *Lecture Notes in Computer Science*, Springer-Verlang, Berlin, Germany, 1997, Vol. 1291, pp. 238-261.

[4] H.D. Karatza, “Gang Scheduling in a Distributed System with Processor Failures”, In *Proceedings of the UK Performance Engineering Workshop*, University of Bristol, Bristol, UK, July 22-23, 1999, pp. 199-208.

[5] H.D. Karatza, “Performance Analysis of Gang Scheduling in a Distributed System Under Processor Failures”, International Journal of Simulation: Systems, Science & Technology, UK Simulation Society, Nottingham, UK, 2001, Vol. 2(1), pp. 14-23.

[6] H.D. Karatza, “Gang Scheduling Performance under Different Distributions of Gang Size”, Parallel and Distributed Computing Practices, Nova Science Publishers, Hauppauge, NY, USA, to appear.

[7] P.G. Sobalvarro, and W.E. Weihl, “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, 1995, Vol. 949, pp. 106-126.

[8] M.S. Squillante, F. Wang, and M. Papaefthymioy, “Stochastic Analysis of Gang Scheduling in Parallel and Distributed Systems”, *Performance Evaluation*, Elsevier, Amsterdam, Holland, 1996, Vol. 27&28 (4), pp. 273-296.

[9] F. Wang, M. Papaefthymiou, and M.S. Squillante, “Performance Evaluation of Gang Scheduling for Parallel and Distributed Systems”, In *Job Scheduling Strategies for Parallel Processing, Lecture Notes in Computer Science*, Springer-Verlang, Berlin, Germany, 1997, Vol. 1291, pp. 184-195.