Performance Analysis of a Distributed System under Time-Varying Workload and Processor Failures

Helen Karatza

Department of Informatics, Aristotle University
54124 Thessaloniki, Greece
Email: karatza@csd.auth.gr

Abstract. This paper studies 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. The distribution for the number of parallel tasks per job varies with time. Two 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. System performance is analysed for different workload parameters.

1 Introduction

Distributed systems have been the focus of research for many years. They consist of several, loosely interconnected processors, where jobs to be processed are in some way apportioned among the processors, and various techniques are used to coordinate processing. However, it is still not clear how to efficiently schedule parallel jobs. To determine this, it is critical to properly assign the tasks to processors and then schedule execution on distributed processors. Good scheduling policies can maximize system and individual application performance, and avoid unnecessary delays.

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. Simulation models are used in this paper 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.

This paper studies and analyzes the performance of different scheduling algorithms in the case where 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. We employed this distribution because in real systems the variability in job parallelism can vary during the day depending on the jobs that run on different time intervals. Also, we compare the performance of different scheduling policies for various coefficients of variation of the processor service times and for different degrees of multiprogramming.

Gang scheduling is studied extensively in [1, 2, 3, 4, 5, 6]. However, these works do not consider processor failures. [7, 8] study gang scheduling under processor failures, but they consider only the uniform distribution for the number of parallel tasks per job (gang size). Furthermore, [7, 8] study smaller degrees of multiprogramming than those examined here. The routing policy that is employed in [7] is based on the shortest queue criterion, while the routing policy that is employed in [8] is based on probabilities. Time-varying distribution for the gang size is considered in [9, 10, 11]. From these papers, [9, 10] do not consider processor failures and the systems that they study are shared memory partitionable parallel systems where all jobs share a single queue. Paper [11] studies a distributed system under processor failures and considers uniform, normal and time varying distributions for the number of tasks per job. This paper is an extension of [11]. It presents and analyzes additional results for the varying with time distribution case, and it provides a more detailed study of the impact on performance of various workload parameters.

The structure of the paper is as follows. Section 2.1 specifies system and workload models, section 2.2 describes scheduling policies, section 2.3 presents the metrics employed in assessing the performance of the scheduling policies that are studied, while model implementation and input parameters are described in section 2.4. The results of the simulation experiments are presented and analyzed in section 3. Section 4 is the conclusion and provides suggestions for further research, and the last section is references.

2 Model and Methodology

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

2.1 System and Workload Models

A closed queuing network model of a distributed system is considered (Fig. 1). There are $P = 16$ homogeneous and independent processors each equipped with its own queue (memory). 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 I/O subsystem has the same service capacity as the processors since we are interested in a system with a balanced program flow. 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.

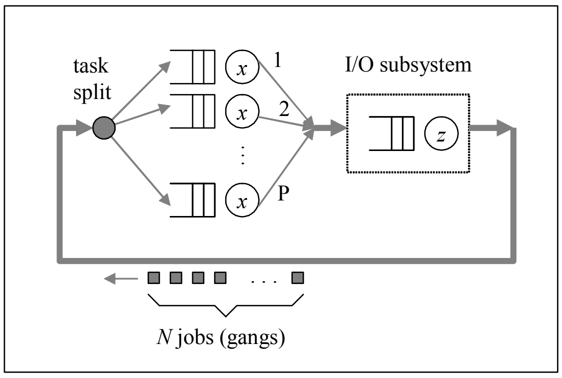

Fig. 1. The queueing network model

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 of job j is represented as $t(j)$. If $p(j)$ represents the number of processors required by job j, then it holds that $1 \leq t(j) = p(j) \leq P$.

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 number of jobs that can be processed in parallel depends on the job size, number of operable processors and the scheduling policy that is employed. The workload considered here is characterized by the following: Distribution of gang sizes, distribution of task service demand, distribution of I/O service time, distribution of processor failures, distribution of processor repair time, and degree of multiprogramming.

Exponentially Time-Varying Distribution of Gang Size. 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 (shown in Fig. 2). 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*.

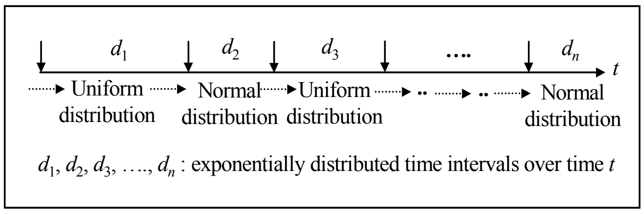

Fig. 2. Exponentially time-varying distribution of gang size

In the uniform distribution case 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 $\eta = (1+P)/2$. In the normal distribution case 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 $\sigma = \eta/4$. 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 as compared with the number of processors.

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 exponential 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*.

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*.

Distributions of Processor Failures and Repair Time. 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 $1/\beta$.

2.2 Scheduling Strategies

We present two well known queuing policies when gang scheduling is used. We assume that the scheduler has correct information about jobs size, i.e. it knows the exact number of processors required by all jobs in the queues.

Adaptive First-Come-First-Served (AFCFS). This policy 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 strategy 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. 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.

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

2.3 Performance Metrics

We consider the definitions: R*esponse 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. *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.

Parameter	Definition
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
D_R	The relative (%) increase in R when we compare a scheduling policy with AFCFS(B)

Table 1. Notations

2.4 Model Implementation and Input Parameters

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 mean time interval for gang size 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.

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 0.10, which means mean repair times (or $1/\beta$) are set to 50, and 100.

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.

3 Simulation Results and Performance Analysis

Tables 2-7 show the mean processor utilization range for all *N* in all cases that we examined. The relative increase in *R* when each of LGFS(B), AFCFS, and LGFS is employed instead of AFCFS(B) is depicted in Figures 3-20. Figures 3-11 represent D_R versus *N* in the $\varphi = 0.05$ case. Figures 12-20 show D_R versus *N* in the φ=0.10 case.

Scheduling policy	$d = 10$	$d = 20$	$d = 30$
AFCFS(B)	0.610-0.650	0.610-0.651	0.609-0.650
LGFS(B)	0.625-0.687	0.624-0.688	0.624-0.688
AFCFS	0.624-0.669	0.624-0.669	0.623-0.670
LGFS	0.639-0.709	0.639-0.709	0.639-0.709

Table 2. *U* range, $C = 1$, $\varphi = 0.05$

Scheduling policy	$d = 10$	$d = 20$	$d = 30$
AFCFS(B)	0.587-0.628	0.586-0.629	0.586-0.629
LGFS(B)	0.600-0.663	0.600-0.664	0.600-0.664
AFCFS	0.608-0.665	0.608-0.665	0.607-0.665
LGFS	0.624-0.705	0.624-0.705	0.623-0.705

Table 3. *U* range, $C = 1$, $\varphi = 0.10$

Scheduling policy	$d = 10$	$d = 20$	$d = 30$
AFCFS(B)	0.599-0.640	0.599-0.640	0.599-0.640
LGFS(B)	0.607-0.673	0.607-0.673	0.606-0.673
AFCFS	0.612-0.658	0.611-0.658	0.611-0.658
LGFS	0.621-0.692	0.621-0.693	0.620-0.693

Table 4. *U* range, $C = 2$, $\varphi = 0.05$

Scheduling policy	$d = 10$	$d = 20$	$d = 30$
AFCFS(B)	0.577-0.619	0.576-0.619	0.576-0.620
LGFS(B)	0.603-0.650	0.584-0.649	0.584-0.650
AFCFS	0.597-0.654	0.596-0.654	0.596-0.654
LGFS	0.607-0.689	0.606-0.689	0.606-0.689

Table 5. *U* range, $C = 2$, $\varphi = 0.10$

Scheduling policy	$d = 10$	$d = 20$	$d = 30$
AFCFS(B)	0.570-0.626	0.569-0.626	0.569-0.626
LGFS(B)	0.573-0.649	0.572-0.649	0.572-0.649
AFCFS	0.581-0.643	0.581-0.643	0.580-0.642
LGFS	0.584-0.667	0.584-0.667	0.583-0.667

Table 6. *U* range, $C = 4$, $\varphi = 0.05$

Scheduling policy	$d = 10$	$d = 20$	$d = 30$
AFCFS(B)	0.549-0.607	0.549-0.605	0.548-0.606
LGFS(B)	0.552-0.628	0.551-0.628	0.551-0.628
AFCFS	0.568-0.639	0.567-0.638	0.567-0.639
LGFS	0.571-0.663	0.570-0.663	0.570-0.663

Table 7. U range, $C = 4$, $\varphi = 0.10$

The results show that with each scheduling method, the mean processor utilization is slightly higher in the non-blocking case than in the blocking case. However, in both cases part of the processor utilization is repeat work caused by processor failures rather than useful work.

In the results presented in Figures 3-20, we observe that the relative performance of the different policies is not significantly affected by the length of the mean time interval for gang size distribution change. The results presented in these Figures also show that the relative performance of the scheduling algorithms depends on φ.

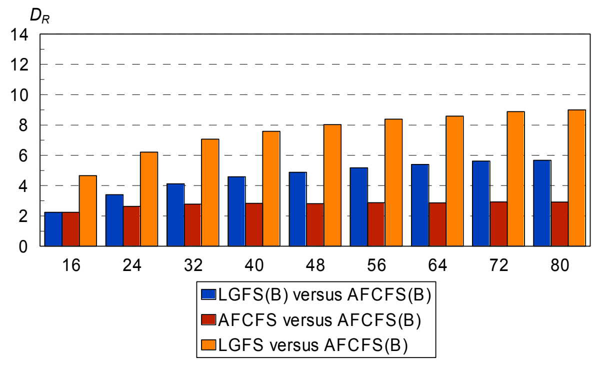

Fig. 3. D_R versus N, $d = 10$, $C = 1$, $\varphi = 0.05$

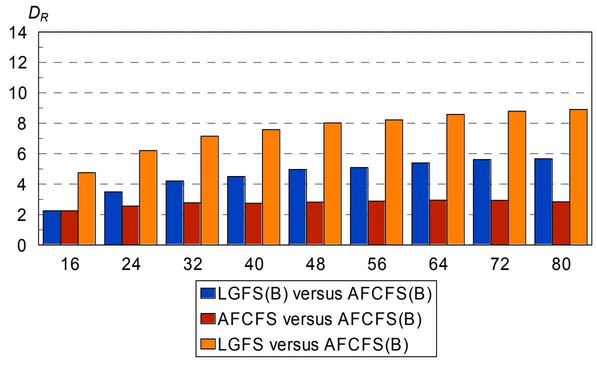

Fig. 4. D_R versus N, $d = 20$, $C = 1$, $\varphi = 0.05$

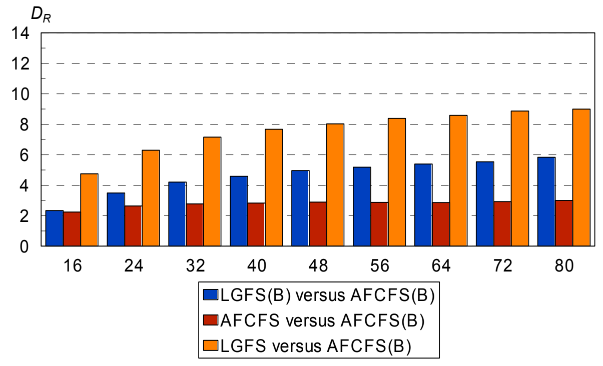

Fig. 5. D_R versus N, $d = 30$, $C = 1$, $\varphi = 0.05$

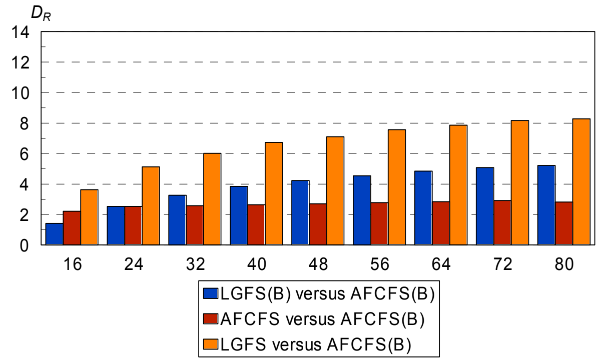

Fig. 6. D_R versus N, $d = 10$, $C = 2$, $\varphi = 0.05$

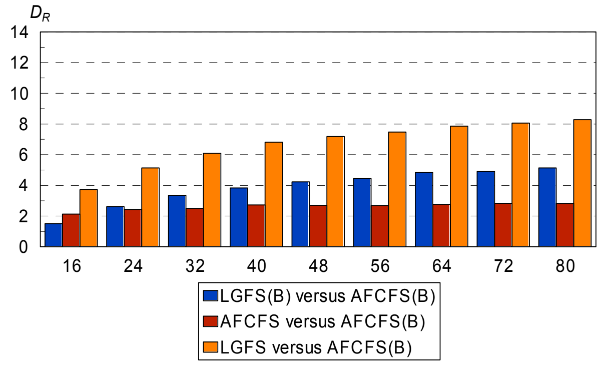

Fig. 7. D_R versus N, $d = 20$, $C = 2$, $\varphi = 0.05$

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 larger (smaller) system throughput.

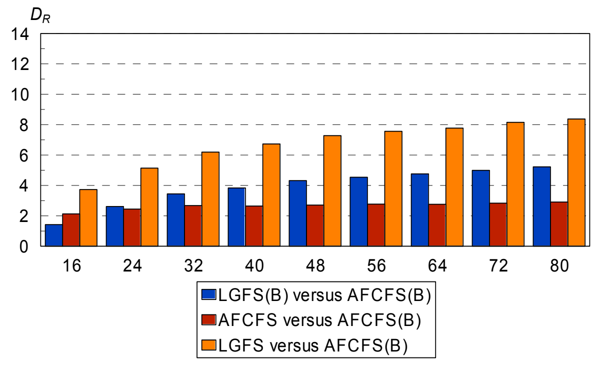

Fig. 8. D_R versus N, $d = 30$, $C = 2$, $\varphi = 0.05$

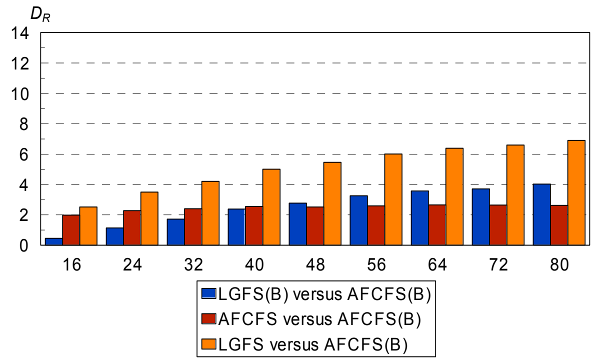

Fig. 9. D_R versus N, $d = 10$, $C = 4$, $\varphi = 0.05$

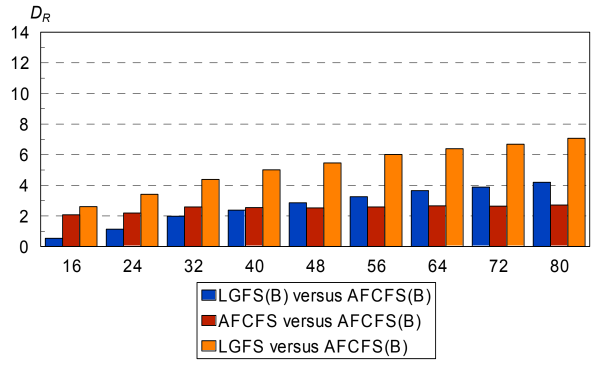

Fig. 10. D_R versus N, $d = 20$, $C = 4$, $\varphi = 0.05$

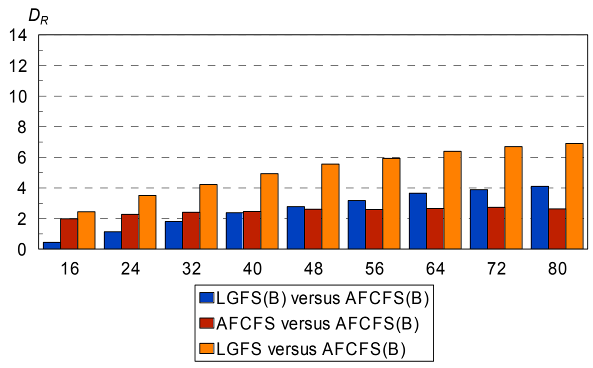

Fig. 11. D_R versus N, $d = 30$, $C = 4$, $\varphi = 0.05$

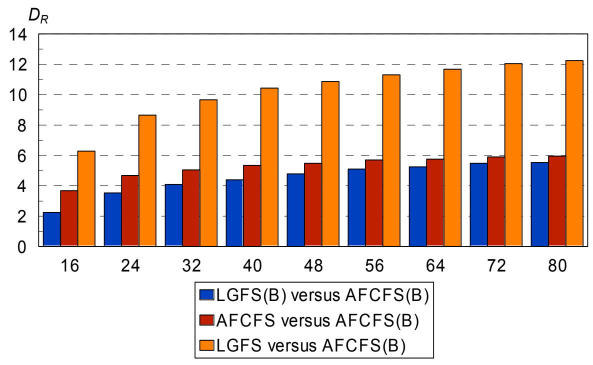

Fig. 12. D_R versus N, $d = 10$, $C = 1$, $\varphi = 0.10$

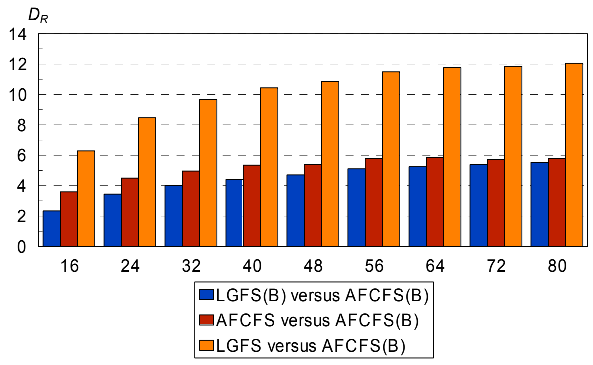

Fig. 13. D_R versus N, $d = 20$, $C = 1$, $\varphi = 0.10$

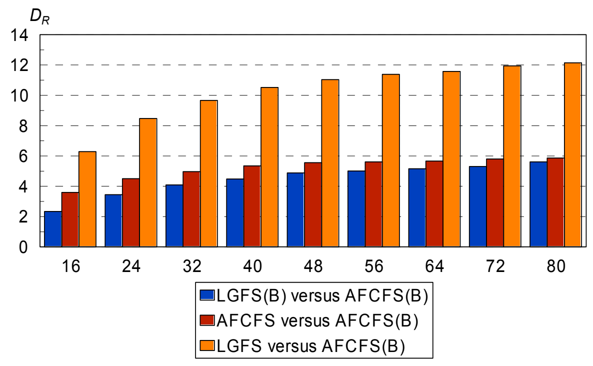

Fig. 14. D_R versus N, $d = 30$, $C = 1$, $\varphi = 0.10$

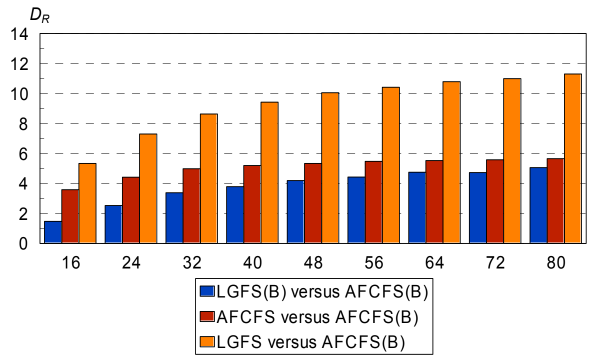

Fig. 15. D_R versus N, $d = 10$, $C = 2$, $\varphi = 0.10$

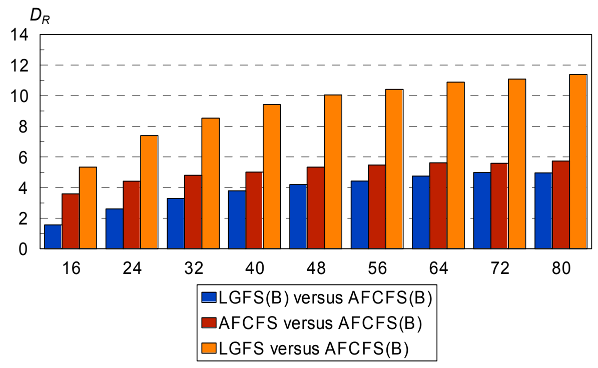

Fig. 16. D_R versus N, $d = 20$, $C = 2$, $\varphi = 0.10$

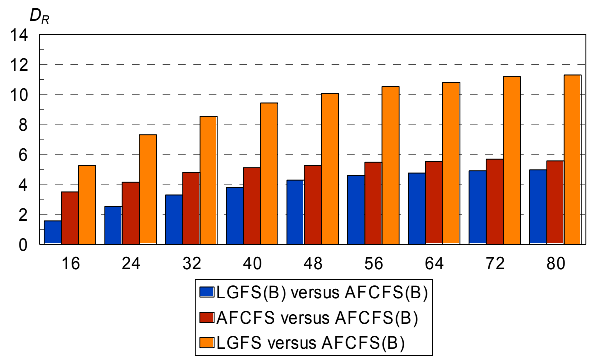

Fig. 17. D_R versus N, $d = 30$, $C = 2$, $\varphi = 0.10$

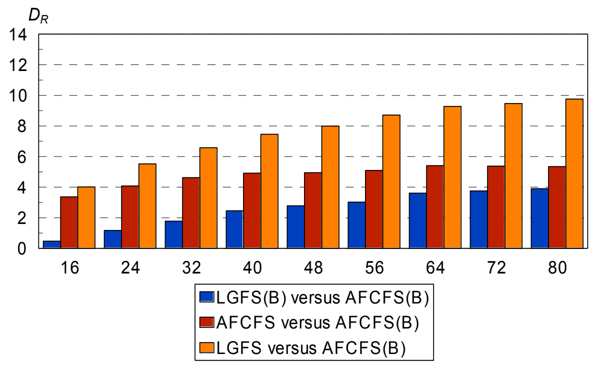

Fig. 18. D_R versus N, $d = 10$, $C = 4$, $\varphi = 0.10$

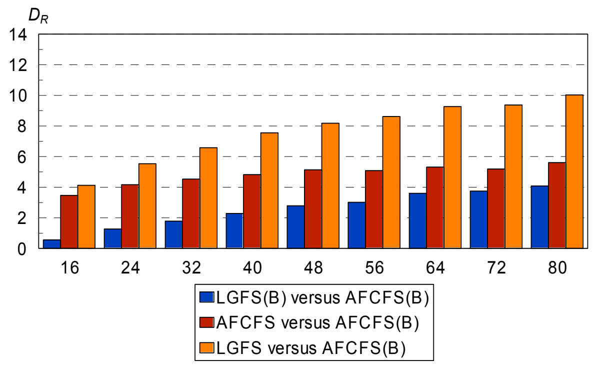

Fig. 19. D_R versus N, $d = 20$, $C = 4$, $\varphi = 0.10$

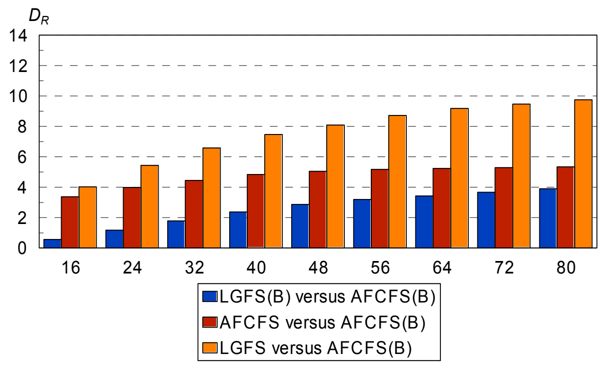

Fig. 20. D_R versus N, $d = 30$, $C = 4$, $\varphi = 0.10$

Generally, the superiority of LGFS(B), LGFS, and AFCFS over AFCFS(B) increases with increasing N (that is, D_R generally increases 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.

D_R generally decreases with increasing C. This is due to the following two facts: a) the variability in gang execution time increases with increasing C, and b) the scheduling methods that we employ do not use criteria that are based on gang service time requirements. Gangs with very long execution times cause anyway long delays to subsequent gangs in the queues independently of the scheduling method that we employ.

D_R is generally larger for $\varphi = 0.10$ than for $\varphi = 0.05$. This is because the advantages of the scheduling methods can be more effective when the repair time is large than when it is smaller.

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, for $\varphi = 0.10$ in all cases AFCFS outperforms LGFS(B). In this case, the difference in performance between these two methods increases with increasing C, but for each C the difference in performance decreases with increasing N. For $\varphi = 0.05$, at $N = 16$ AFCFS either outperforms LGFS(B) ($C > 1$) or performs very close to LGFS(B) ($C = 1$). As N increases, the difference between these two methods first tends to zero and then from some N that depends on C, LGFS(B) outperforms AFCFS with a difference that generally increases with increasing N. The reason that the N where the relative performance of these two methods changes is different at different C, is because for each N, the mean processor utilization is lower at larger C.

4 Conclusions and Future Research

The simulation results show that the LGFS method outperforms the other methods. Furthermore, the results show that the relative performance of the scheduling policies depends on the variability of processor service times, the degree of multiprogramming, and the failure to repair ratio. We also conclude that the length of gramming, and the failure to repair ratio. We also conclude that the length of the mean time interval for the gang size distribution change does not significantly affect performance. In this paper we studied processor failures in a closed queueing network model of a distributed system where the number of jobs is constant during a simulation experiment. As a future research we plan to consider an open queueing network model and to study the impact on performance of a time-varying distribution for job inter-arrival times.

References

1. Feitelson D.G., Rudolph R.: Parallel Job Scheduling: Issues and Approaches. In: Feitelson, D.G., Rudolph, R. (eds.): *Job Scheduling Strategies for Parallel Processing*. Springer LNCS Vol.949. (1995) 1-18
2. Feitelson D.G., Rudolph R.: Evaluation of Design Choices for Gang Scheduling Using Distributed Hierarchical Control. *Journal of Parallel and Distributed Computing*, 35:18-34, (1996)
3. Feitelson D.G., Jette M.A.: Improved Utilisation and Responsiveness with Gang Scheduling". In: Feitelson, D.G., Rudolph, R. (eds.): *Job Scheduling Strategies for Parallel Processing*. Springer LNCS, Vol.1291. (1997) 238-261
4. Sobalvarro P.G., Weihl W.E.: Demand-Based Coscheduling of Parallel Jobs on Multiprogrammed Multiprocessors. In: Feitelson, D.G., Rudolph, R. (eds.): *Job Scheduling Strategies for Parallel Processing*. Springer LNCS, Vol.949, (1995) 106-126
5. Squillante M.S., Wang, F., Papaefthymioy, M.: Stochastic Analysis of Gang Scheduling in Parallel and Distributed Systems. *Performance Evaluation*, 27-28(4):273-296 (1996)
6. Wang F., Papaefthymiou M., Squillante M.S. Performance Evaluation of Gang Scheduling for Parallel and Distributed Systems. In: Feitelson, D.G., Rudolph, R. (eds.): *Job Scheduling Strategies for Parallel Processing*. Springer LNCS, Vol.1291. (1997) 184-195
7. Karatza H.D.: Gang Scheduling in a Distributed System with Processor Failures. *Proceedings UK Performance Engineering Workshop*. University of Bristol, Bristol (1999) 199-208
8. Karatza H.D.: Performance Analysis of Gang Scheduling in a Distributed System Under Processor Failures. *International Journal of Simulation: Systems, Science & Technology*, 2 (1):14-23 (2001)
9. Karatza H.D.: Gang Scheduling Performance under Different Distributions of Gang Size. *Parallel and Distributed Computing Practices*, 4(4):433-449 (2003)
10. Karatza H.D.: Co-Scheduling in a Shared-Memory Multiprocessor System Under Time Varying Workload. *Proceedings 6th UK Simulation Society Conference*. Cambridge (2003) 181-187
11. Karatza H.D.: Gang Scheduling in a Distributed System under Processor Failures and Time-Varying Gang Size. *Proceedings 9th IEEE Workshop on Future Trends of Distributed Computing Systems*. Puerto Rico (2003) 330-336