A Simulation-Based Performance Analysis of Gang Scheduling in a Distributed System

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

Abstract

In distributed systems job scheduling is a difficult problem. In this work we study a special type of scheduling called gang scheduling under which jobs consist of a number of interacting tasks which are scheduled to run simultaneously on distinct processors. The performance of various gang scheduling schemes is studied and compared for a variety of workloads. The main objective of the processor schedulers is to achieve high overall system throughput while at the same time providing some guarantee for the performance of individual jobs in the system.

1. Introduction

In distributed systems there is a growing need for good schedulers that will manage the distributed nodes. However, it is not obvious how to partition the computing nodes of the distributed system among competing jobs. One idea, is *gang scheduling*, that is the case where a set of tasks are scheduled to execute simultaneously on a set of processors. It allows the tasks to interact efficiently by using busy waiting, without the risk of waiting for a task that currently is not running. Without gang scheduling, tasks have to block in order to synchronise, thus suffering the overhead of a context switch.

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

At any time there is a one-to-one mapping between tasks and processors. We emphasise that although the total number of tasks in the system may be larger that the number of processors, no gang contains more tasks than 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 ([2], [9]), in which, after a job arrives to the system, it is immediately split into component tasks, and these tasks will be processed on any processor in any order as long as the precedence constraints are not violated.

A number of gang scheduling policies for distributed systems and multiprogrammed parallel systems have been proposed already, each differing in the way resources are shared among the jobs ([1], [3], [4], [5], [6], [7], [8], [13], [14], [15]). These works study the influence of gang scheduling on processors performance. They do not explicitly model the I/O processing, although it can significantly influence the overall system performance. However, scheduling is not an isolated issue. It is but one service provided by the operating system. The solution to the scheduling problem must be integrated with solutions to other problems, e.g. I/O management. The different parts of the system must work together to create a cohesive whole in a way that makes sense.

Coscheduling of tasks in a closed queueing network model is studied in [10]. A shared memory partitionable parallel processing system is considered in this work and resequencing of jobs is required after processor service. Eager (work-conserving) versus lazy (non-work-conserving) scheduling policies are studied and compared over a wide range of system parameters.

Numerous scheduling disciplines have been proposed for multiprocessor systems, the evaluation of which, for the most part, was conducted on workloads with a relatively small variability in task processing requirements. However, high performance computer centers have report that their service time coefficient of variation can in fact be greater than one.

In this work we study gang scheduling in a closed queueing network model of a distributed system where we

incorporated I/O equipment. The design choices that are considered include different ways to schedule gangs for execution. We compare the performance of three gang scheduling policies for various coefficients of variation of the processor service times and for different degrees of multiprogramming. To our knowledge, such an analysis of gang scheduling has not appeared in the research literature.

This paper is theoretical in the sense that the results are obtained from simulation studies instead of from measurements of real systems. Nevertheless, we believe that the results we present are of practical value. All algorithms we study are practical in that they can be implemented. Although we do not derive absolute performance predictions for specific systems and workloads, we study the relative performance of the different gang scheduling algorithms across a broad range of workloads and analyze how changes in the workload affect performance.

For simple idealized systems, performance models can be mathematically analyzed using queueing theory to obtain performance measures. Our system, in addition to exponential distribution for job execution times, involves Branching Erlang and Erlang-k distributions. Also, it involves scheduling policies with different complexities. For complex systems analytical modelling typically requires additional simplifying assumptions, and such assumptions might have unforseeable influences on the results. Therefore, research efforts have been devoted to finding approximate analysis, to developing tractable models in special cases, and to conducting simulations. We chose simulations because it is possible to simulate the system under study in a direct manner, thus lending credibility to the results. Detailed simulation models help determine performance bottlenecks in architecture and assist in refining the system configuration.

The structure of the paper is as follows. Section 2.1 specifies system and workload models, section 2.2 describes the job scheduling policies and section 2.3 presents the metrics employed in assessing the performance of the scheduling policies we study. Model implementation and input parameters are described in section 3.1 while the results of the simulation experiments are presented and analysed in section 3.2. Finally the last section concludes with several comments and gives directions for further research.

2. Model and methodology

2.1 System and workload models

A closed queuing network model of a distributed system is considered. There are $P = 8$ homogeneous and independent processors each serving its own queue. This is a reasonable choice for the current existing medium-scale departmental networks of workstations. It is believed that qualitative results for other numbers of processors, even for large-scale distributed, are similar for the data demonstrated here. The distributed nodes are interconnected by a high speed network with negligible communication delays.

The degree of multiprogramming N is constant during the simulation experiment. A fixed number of jobs N is circulating alternatively between the processors and the I/O unit. Neither arrivals nor departures are permitted while the system is under observation. The configuration of the model is shown in Figure 1.

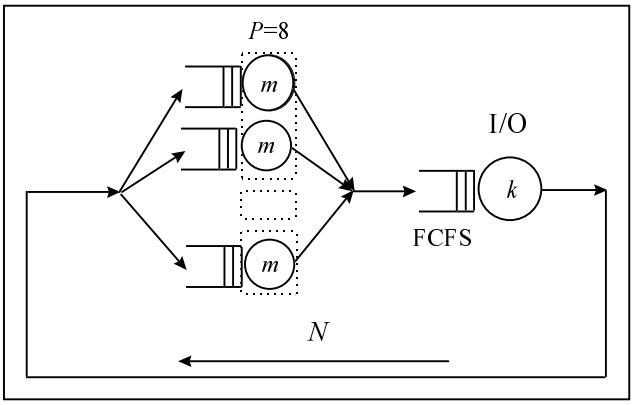

Figure 1. The queuing network model.

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

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

The workload considered here is characterised by three parameters: the distribution of gang sizes, the distribution of execution times, and the degree of multiprogramming. We assume that there is not correlation between the different parameters. For example, a small gang may have a long execution time.

Jobs consist of a set of $n \geq 1$ tasks and each task requires one processor to execute. Jobs are characterised by a probability distribution of the number of processors required. The number of processors demanded by job x is represented as $p(x)$, which is called the "size" of job x. We

call a job "small" ("large") if it requires a small (large) number of processors respectively. Each time a job returns from I/O service to distributed processors, it needs 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 x starts to execute only if all $p(x)$ processors assigned to it are available. Otherwise, all tasks of job x wait in the assigned queues. When a job finishes execution, all processors assigned to it are released. Jobs x_1, x_2, ... x_l can be executed simultaneously if and only if $p(x_1)+p(x_2)+.....+p(x_l) <= P$.

We assume that job sizes are uniformly distributed in the range of [1..8]. We have chosen the uniform distribution because it has a reasonable large deviation and therefore more facts about system behaviour can be revealed. The number of jobs which can be processed in parallel depends on job sizes and on the scheduling policy applied. As in most studies we assume that the overhead for the scheduling algorithms is negligible.

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

The parameter which represents the variability in job execution times is the coefficient of variation of execution times (C). This is the ratio of the standard deviation of job execution time to its mean. We examine the following cases:

- Job execution times have an Erlang-k distribution with k=2 and are IID. The coefficient of variation is $C = 1/\sqrt{k} < 1$ and the mean is m.
- Job execution times are independent and identically distributed (IID) exponential random variables with mean m.
- Job execution times have a Branching Erlang distribution ([12]) with two stages and are IID. The coefficient of variation is C, where $C > 1$ and the mean is m.

A job after processor service requests service from the I/O unit. The I/O queuing discipline is FCFS. The I/O service times are exponentially distributed with mean k and are IID.

All notations used in this paper are presented in Table 1.

2.2 Job scheduling policies

The performance of programs which consist of parallel tasks is significantly affected by the choice of the policy used to schedule tasks. In this work we assume that the scheduler has perfect information when making decisions, i.e. it knows the exact number of processors required by all jobs. Also, the task dispatcher knows the exact length of all queues and sends tasks to the shortest queues, one task per queue.

We analyse the performance of the following gang scheduling policies:

- *First Come First Served* (*FCFS*). When a job after I/O service asks processor service, then: if all processors assigned to it are idle and there are not any other tasks in these processors queues, this job starts execution. Otherwise all tasks of that job enter processor queues.

When a job x leaves the system, the $p(x)$ processors assigned to it are released. Then the queues of these processors, and also the queues of the idle processors are examined. If there are jobs whose all tasks are in the first position in these queues, then these jobs start execution. A job whose at least one task is not in the first position of a queue, does not start execution.

Unfortunately, FCFS scheduling by its nature is conservative in scheduling. Tasks may be kept in queues even if the corresponding processors are idle. We can modify the FCFS scheduling algorithm so that small jobs are processed early instead of letting them waiting behind large jobs.

- *Adapted FCFS* (*A-FCFS*) attempts to schedule a job whenever the processors assigned to its tasks are available. When there are no enough processors available for a large job whose tasks are waiting in the front of the queues, A-FCFS policy schedules smaller jobs whose tasks are behind the tasks of the large job, so that the average response time could be reduced.

One major problem with this scheduling policy is that it tends to favour those jobs requesting a smaller number of processors and thus may increase the fragmentation in the system.

- *Largest-Job-First-Served* (*LJFS*). With this policy tasks are placed in increasing job size order in processor queues (tasks that belong to larger gangs are placed ahead in the queues). All tasks in queues are searched in order, and the first jobs for which the assigned processors are available

start 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 have a mandate to run large, highly-parallel jobs that cannot run anywhere else.

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

RT	mean response time
K	mean cycle time
R	system throughput rate
U_{CPU}	mean processor utilisation
$U_{I/O}$	mean I/O unit utilisation
N	degree of multiprogramming
C	coefficient of variation
m	mean job execution time
k	mean I/O service time

In our model the external performance is determined by the system throughput rate (system performance) and the mean cycle time (program performance). Internal efficiency is primarily determined by the mean processor utilization because it represents the level of contention for the most critical system resources.

FCFS is used as a comparison measure between the performance of the A-FCFS and LJFS policies. The model works first with the FCFS policy and then with one of the other two policies. The relative performance parameters are calculated on a percentage basis as follows:

D_{RT} : relative decrease in RT

D_K : relative decrease in K

D_R : relative increase in R

3. Simulation results and discussion

3.1 Model implementation and input parameters

The queuing network model was simulated with discrete event simulation models ([11]) using the independent replications method. For every mean value a 95% confidence interval was evaluated. All confidence intervals were found to be less than 5% of the mean values.

We considered a balanced system with m=1.0 and k = 0.563. The reason we have chosen k=0.563 for balanced program flow is that at the processors there are on average 4.5 parallel tasks per job. So, when all processors are busy, an average of 1.778 jobs are served each unit of time. This implies that I/O mean service time must be equal to 1/1.778 = 0.563 if the I/O unit is to have the same service capacity.

The system was examined in cases of job execution times with Erlang-k distribution with k=2 ($C = 1/\sqrt{2} = 0.707$), exponential (C = 1), and Branching Erlang for C = 2, 4.

The degree of multiprogramming N was taken as 2, 4, 6, 8, 10. The reason we have examined various numbers of programs N is because this is a critical parameter which reflects system load.

3.2 Performance analysis

Due to space limitations, only the following results are presented, but they are representative of the overall model performance:

- In Tables 2-6 performance parameters of the C=1 case are presented.

- In Figures 2-9 system throughput rate and mean cycle time are plotted versus N in all cases examined.

Our results show the following:

For N=2 all policies perform the same as there can be at most one job waiting for processor service.

For N>2 the performance in terms of mean cycle time (K) and throughput rate (R), is superior with the A-FCFS and LJFS scheduling policies. This is due to the fact that with the FCFS policy a large job waiting for some processors to be freed may cause long delays to other smaller jobs. During that time it is most probable for the I/O unit to remain idle and then to be supplied with many jobs that are forced to delay in its queue. A-FCFS and LJFS alleviate

this problem, yielding lower *RT* than FCFS, which results in lower mean cycle time and higher system throughput.

In most of the cases LJFS performs slightly better than A-FCFS. However, LJFS needs an extra overhead for the reordering of tasks in the queues which is not modelled in this work.

The superiority of A-FCFS and LJFS over FCFS is increasing with increasing *N*. This is due to the fact that at high *N* there are more tasks in the queues and therefore there are more opportunities to exploit the advantages of A-FCFS and LJFS. For low *N* the superiority is not significant while for *N*=10 is considerable. For example with LJFS and for *N*=4 we have D_R =3.45% while for *N*=10 we have D_R = 16.83%.

For all *N*, the difference in performance between FCFS and each of A-FCFS and LJFS is decreasing with increasing *C*. This is due to the fact that as *C* increases the variability in job execution time increases too. Therefore at high *C* the probability for large jobs (jobs with many tasks) to cause long delays to smaller jobs is small because it is most probable for service times to be very small. So, at high *C* the advantages of A-FCFS and LJFS are not completely exploited.

4. Conclusions and further research

In this work we studied gang scheduling on a distributed system. We used simulation as the means of obtaining results.

Three gang scheduling policies were considered. First Come First Served (FCFS), Adapted-FCFS (A-FCFS), and Last-Job-First-Served (LJFS). Their performance was studied and compared for various degrees of multiprogramming *N* and coefficients of variation *C* of task execution times.

The simulation results reveal the following:

- In all cases examined A-FCFS and LJFS performed better than FCFS.
- A-FCFS performed very close to LJFS. Actually, in most of the cases LJFS performed only slightly better than A-FCFS.
- The superiority of A-FCFS and LJFS over FCFS is increasing with increasing *N* and is decreasing with increasing *C*.

Both A-FCFS and LJFS policies have their merits. System fragmentation can be reduced with LJFS. However, there is an overhead involved with LJFS, due to reordering of tasks in the queues which is not modelled in this work. For these reasons we believe that A-FCFS method should be used as it is easier to implement and it performs very close to LJFS.

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

- Different job size distributions to be considered at larger distributed systems.
- System behaviour to be investigated in the presence of processor failures.

Table 2. *C*=1, FCFS case

N	U_{CPU}	$U_{I/O}$	*RT*	*K*	*R*
2	0.529	0.531	1.342	2.099	0.953
4	0.617	0.618	2.483	3.606	1.109
6	0.637	0.633	3.890	5.281	1.136
8	0.636	0.638	5.459	6.990	1.145
10	0.637	0.633	7.268	8.799	1.136

Table 3. *C*=1, A-FCFS case

N	U_{CPU}	$U_{I/O}$	*RT*	*K*	*R*
2	0.529	0.531	1.342	2.099	0.953
4	0.645	0.640	2.387	3.485	1.148
6	0.684	0.685	3.477	4.884	2.229
8	0.707	0.701	4.776	6.364	1.257
10	0.723	0.719	5.977	7.752	1.290

Table 4. *C*=1, LJFS case

N	U_{CPU}	$U_{I/O}$	*RT*	*K*	*R*
2	0.529	0.531	1.342	2.099	0.953
4	0.641	0.640	2.362	3.486	1.147
6	0.694	0.687	3.441	4.868	1.232
8	0.716	0.711	4.588	6.275	1.275
10	0.739	0.740	5.672	7.532	1.328

Table 5. *C*=1, FCFS versus A-FSCS

N	D_{RT}	D_K	D_R
2	0.00	0.00	0.00
4	3.86	3.36	3.47
6	10.63	7.52	8.14
8	12.50	8.95	9.83
10	17.76	11.90	13.51

Table 6. *C*=1, FCFS versus LJFS

N	D_{RT}	D_K	D_R
2	0.00	0.00	0.00
4	4.85	3.33	3.45
6	11.54	7.82	8.48
8	15.96	10.23	11.39
10	21.95	14.41	16.83

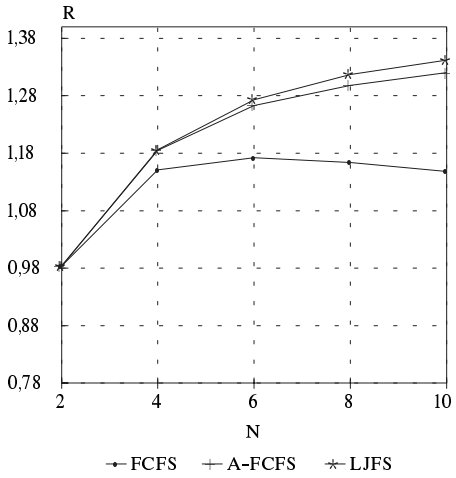

Figure 2. *R* versus *N*, *C*=0.707

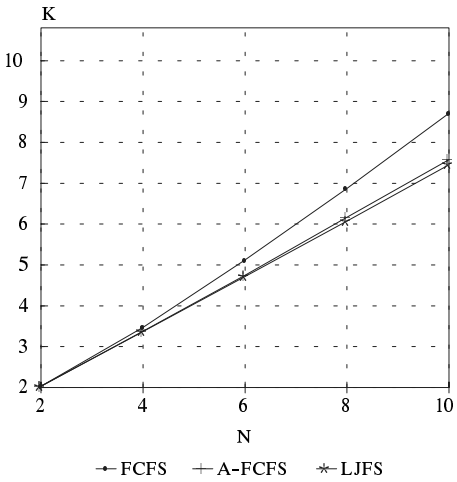

Figure 3. *K* versus *N*, *C*=0.707

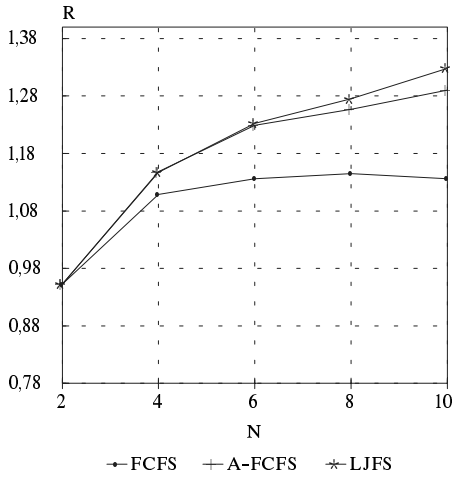

Figure 4. *R* versus *N*, *C*=1

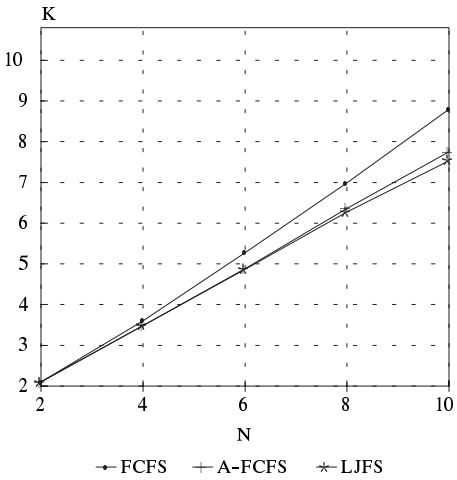

Figure 5. *K* versus *N*, *C*=1

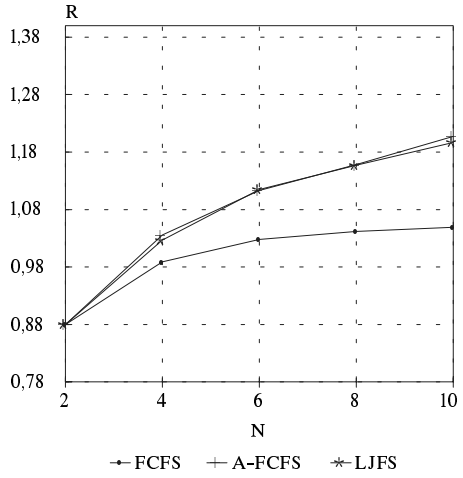

Figure 6. *R* versus *N*, *C*=2

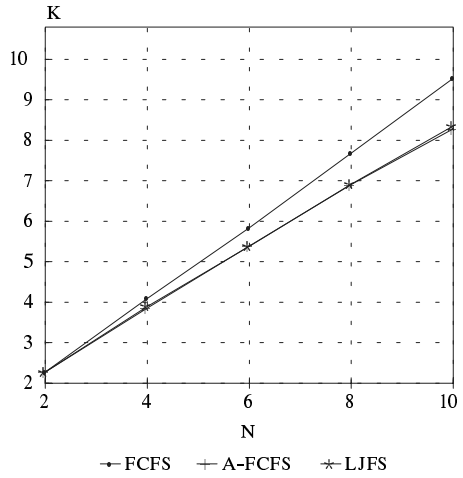

Figure 7. *K* versus *N*, *C*=2

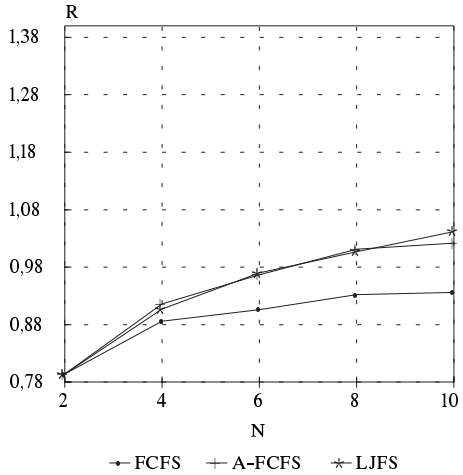

Figure 8. *R* versus *N*, *C*=4

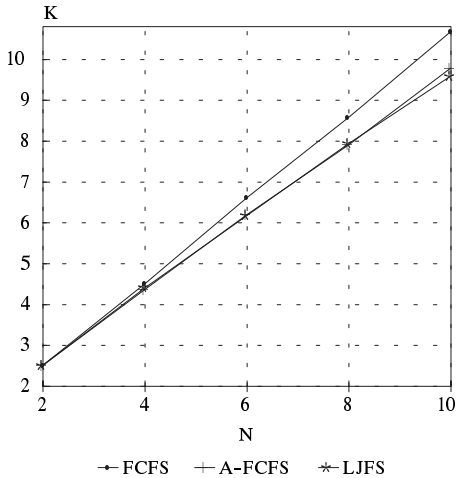

Figure 9. *K* versus *N*, *C*=4

References

[1] M.J. Atallah, C. L. Black, D. C. Marinescu, H. J. Siegel, and T. L. Casavant, "Models and Algorithms for Coscheduling Compute-Intensive Tasks on a Network of Workstations". *Journal of Parallel and Distributed Computing*, Vol. 16, 1992, pp. 319-327.

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

[3] D.G. Feitelson, and L. Rudolph, "Gang Scheduling Performance Benefits for Fine-Grain Synchronization". *Journal of Parallel and Distributed Computing,* Vol. 16, 1992, pp. 306-318.

[4] D.G. Feitelson, and L. Rudolph, "Parallel job scheduling: issues and approaches". In *Job Scheduling Strategies for Parallel Processing*, D.G. Feitelson and L. Rudolph (eds.), Springer-Verlang, Lecture Notes in Computer Science Vol. 949, 1995, pp. 1-18.

[5] D.G. Feitelson, and L. Rudolph, "Evaluation of Design Choices for Gang Scheduling Using Distributed Hierarchical Control". *Journal of Parallel and Distributed Computing*, Vol. 35, 1996, pp. 18-34.

[6] D.G. Feitelson, and M. A. Jette, "Improved Utilization and Responsiveness with Gang Scheduling". In *Job Scheduling Strategies for Parallel Processing*, D.G. Feitelson and L. Rudolph (eds.), Springer-Verlang, Lecture Notes in Computer Science, Vol. 1291, 1997, pp. 238-261.

[7] M.A. Jette, "Performance Characteristics of Gang Scheduling in Multiprogrammed Environments". Lawrence

Livermore National Laboratory, http://www-lc.llnl.gov/-global_access /dctg/gang/sc97.paper.html, 1997.

[8] M.A. Jette, “Gang Scheduling, Timesharing on Parallel Computers”. Lawrence Livermore National Laboratory, http: // www - lc.llnl.gov/ global_access/ dctg/ gang/ sc98. summary.html, 1998.

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

[10] H.D. Karatza, “Eager Scheduling versus Lazy Scheduling with Resequencing”. In *Proceedings of the 1998 Symposium on Performance Evaluation of Computer and Telecommunication Systems* (SPECTS '98) - 1998 Summer Computer Simulation Conference (SCSC '98), Reno, Nevada, July 19-22, 1998, pp. 261-267.

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

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

[13] P.G. Sobalvarro, and W. E. Weihl, “Demand-based Coscheduling of Parallel Jobs on Multiprogrammed Multiprocessors”. In *Lecture Notes in Computer Science*, 949, D. G. Feitelson and L. Rudolph (eds.), Job Scheduling Strategies for Parallel Processing, IPPS ’95 Workshop, Santa Barbara, CA, April 1995, pp. 106-126.

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

[15] 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*, D.G. Feitelson and L. Rudolph (eds.), Springer-Verlang, Lecture Notes in Computer Science, Vol. 1291, 1997, pp. 184-195.