Concerning the Length of Time Slots for Efficient Gang Scheduling*

B. B. Zhou and A. M. Goscinski
School of Computing & Mathematics
Deakin University
Geelong, VIC 3217, Australia
bbz,ang@deakin.edu.au

R. P. Brent
Oxford University Computing Laboratory
Wolfson Building, Parks Road
Oxford OX1 3QD, UK
rpb@comlab.ox.ac.uk

Abstract

Applying gang scheduling can alleviate the blockade problem caused by exclusively used space-sharing strategies for parallel processing. However, the original form of gang scheduling is not practical as there are several fundamental problems associated with it. Recently many researchers have developed new strategies to alleviate some of these problems. Unfortunately, one important problem has not been so far seriously addressed, that is, how to set the length of time slot to obtain a good performance of gang scheduling. With gang scheduling time is divided into time slots of equal length, the number of time slots introduced in the system forms a scheduling round and each new job is first allocated to a particular time slot and then starts to run in the following scheduling round. Ideally, the length of time slot should be set long to avoid frequent context switches and so to reduce the scheduling overhead. The number of time slots in a scheduling round should also be limited to avoid a large number of jobs competing for limited resources (CPU time and memory). Long time slots and the limited number of time slots in each scheduling round may cause jobs to wait for a long time before it can be executed after arrival, which can significantly affect the performance of jobs, especially short jobs which are normally expected to finish quickly. However, the performance of a short job can also suffer if the length of time slot is not long enough to let the short job complete in a single time slot. In this paper we present a strategy to deal with this important issue for efficient gang scheduling.

1 Introduction

Scheduling strategies for parallel processing can be classified into either *space sharing* or *time sharing*. Due to its simplicity, currently most commercial parallel systems only adopt space sharing. With space sharing each partitioned processor subset is dedicated to a single job and the job will exclusively occupy that subset until completion. One major drawback of space sharing is the blockade situation, that is, small jobs can easily be blocked for a long time by large ones. The *backfilling* technique can be applied to alleviate this problem to a certain extent [9, 11]. However, the blockade can still be a serious problem under heavy workload. To alleviate this problem, time sharing needs to be considered.

Because the processes of the same parallel job may need to communicate with each other during the computation, in a time-shared environment the execution of parallel jobs should be coordinated to prevent jobs from interfering with each other. Coordinated scheduling strategies can be classified into two different categories. The first is called *implicit coscheduling* [3, 12]. This approach does not use a global scheduler, but local schedulers on each processor to make scheduling decisions mainly based on the communication behavior of local processes. Implicit coscheduling is attractive for loosely coupled clusters without a central resource management system.

The second type of coscheduling is called *explicit coscheduling* [10] or *gang scheduling* [7]. With gang scheduling time is divided into time slots of equal length, the number of time slots introduced in the system forms a scheduling round, and each new job is first allocated to a particular time slot and then starts to run at the following scheduling round. Controlled by a global scheduler, all parallel jobs in the system take turns to receive the service in a coordinated manner. It gives the user an impression that the job is not blocked, but executed on a dedicated slower machine when the system workload is heavy.

Although many new strategies have been introduced to enhance the performance of gang scheduling, one important problem has not been so far seriously addressed, that is, how to set the length of time slot to obtain a good system utilisation and job performance. Ideally, the length of time slot should be set long to avoid frequent context switches and

*To appear in *Proc. PDCAT 2002*, 364–371.

so to reduce the scheduling overhead. The number of time slots in a scheduling round should also be limited to avoid a large number of jobs competing for limited resources (CPU time and memory). Long time slots and the limited number of time slots in each scheduling round may cause jobs to wait for a long time before they can be executed after arrival, which can significantly affect the performance of jobs, especially short jobs which are normally expected to finish quickly. However, the performance of a short job can also suffer if the length of time slot is not long enough to let the short job complete in a single time slot. In this paper we present a strategy to deal with this important issue for efficient gang scheduling.

The paper is organised as follows. First we briefly discuss some related work in Section 2. In Section 3 the new strategy is described. The experimental system and the workloads used in our experiments are discussed in Section 4. Experimental results are presented in Section 5. Finally the conclusions are given in Section 6.

2 Related Work

There are certain fundamental problems associated with the original form of gang scheduling. The first problem is initial allocation of resources to new arrivals to balance the workload across the processors. Various methods of memory allocation, such as *first fit*, *best fit* and *buddy*, can be used for processor and time slot allocation for parallel processing. However, study [4] shows that the buddy system approach performs best if two adjacent time slots are allowed to be unified into a single one for the same job during the computation (a kind of running jobs in multiple time slots). A famous method for implementing the buddy system and also taking the workload balance into consideration is the *distributed hierarchical control*, or DHC for parallel processing [5].

System workload changes in a random manner during the computation due to job arrivals and departures. There is a need for effective methods to reallocate resources to balance the workload and to enhance the system utilisation. One important issue is how freed processors due to job termination in one time slot could be effectively reallocated to existing jobs running in other time slots if there are no new jobs on arrival. One possible solution is to allow jobs to run in multiple time slots whenever possible [4, 13]. However, our study [16] shows that simply running jobs in multiple time slots may not be able to enhance the system performance. This is because long jobs stay in the system longer and thus are more likely to run in multiple time slots. In consequence the performance of short jobs may be degraded as they can only obtain a small amount of CPU time in each scheduling round. Job should be allowed to run in multiple time slots. However, special care has to be taken to prevent the above situation from happening. An effective way of achieving this is to minimise whenever possible the total number of time slots in each scheduling round, while allowing jobs to run in multiple time slots [16].

Another method which can effectively handle both initial allocation and reallocation during the computation is *job re-packing* [16]. For initial allocation this method is similar to DHC method, but enhanced by using a simplified procedure to balance the workload across the processors. A big advantage of this method is that the order of job execution in the system is allowed to change during computation so that processors in the system can be utilised more efficiently. With job re-packing we are able to simplify the search procedure for available processors, to balance the workload across the processors and to quickly determine when a job can run in multiple time slots and when a time slot can be eliminated to minimise the number of time slots in each scheduling round. Experimental results show that, using job re-packing, both processor unilisation and job performance can be greatly enhanced [16].

The computing power of a given system is limited. If many jobs time-share the same set of processors, each job can only obtain a very small portion of processor time and no job can complete quickly. Thus the number of time slots in a scheduling round should be limited. However, the system may work just like a FCFS queuing system if we simply limit the number of time slots in each scheduling round. The question is if the performance can be enhanced by combining gang scheduling with good space sharing strategies. The experimental results presented in [14] show that by adopting the backfilling technique the performance of gang scheduling can indeed be improved when the number of time slots is limited.

To apply backfilling technique we need the information about the length of each individual job. With such information available we can further enhance the system performance by dividing jobs into several classes based on their required service times. Our experimental results show that the improvement of system performance can be significant if we limit the number of long jobs simultaneously running on the same processors [17]. This is because the average number of time slots in a scheduling round will be reduced if less number of long jobs is running on the same processors at the same time, and then the average performance of short jobs is improved as they have better chance to be executed immediately without waiting in a queue. Note the average performance of long jobs can also be enhanced by limiting the number of long jobs simultaneously running on the same processors. This can be proved by comparing the results of average turnaround time for the same set of long jobs running under two different system: the FCFS and the round robin.

3 The Strategy

Our strategy for setting time slots with a reasonable length, but not seriously degrading the performance of short jobs is described in this section.

We shall show in Section 5 that to increase the length of time slots can markedly decrease the performance of short jobs. The reason can be explained as follows. With gang scheduling using the simple round robin strategy, each job is first allocated to a particular time slot and then starts to run at the following scheduling round. If the limit of time slots is reached and the job cannot be allocated in the existing time slots, it has to be queued. Assume that the limit of time slots in a scheduling round is n and the length of time slot is l. The maximum time for a scheduling round is thus nl. If l is long, the time for a job to wait before being executed can be long even if it can be executed in the following scheduling round. The waiting time can be much longer if the job has to wait in the waiting queue when the limit of time slots in the scheduling round is reached. This amount of waiting time can significantly affect the performance of short jobs. However, we should not set the length of time slots short. The reasons are as follows. First, short time slots do not reduce the waiting time a job spends in the waiting queue. Second, setting the length of time slot short dramatically increases the frequency of context switches and so increases the scheduling overhead. Third, the performance of short jobs may greatly be degraded because the execution time for a short job can be proportional to nl if it cannot complete in a single time slot. The question is thus whether we can find a method which is able to set the length of time slot reasonably long and at the same time guarantee the performance of short jobs not to be degraded significantly.

In order to provide a special treatment to short jobs we need to divide jobs into different classes. Conventionally, jobs are not distinguished according to their execution times when gang scheduling is considered. It should be pointed out that the simple round robin scheme used in gang scheduling works well only if the sizes of jobs are distributed in a wide range. Gang scheduling using the simple round robin strategy may not perform as well as even a simple FCFS scheme in terms of average response time, or average slowdown, when all the incoming jobs are large. As discussed in the previous section, the classification of jobs based on job length is achievable because the backfilling technique requires the information of service time of each individual job. In our previous research [17] we demonstrated that by limiting the number of long jobs simultaneously running on the same processors, the performance of both long and short jobs are improved. (A similar result using a different workload model will be presented in Section 5).

It seems that setting the limit for the number of simultaneously running long jobs has nothing to do with the length of time slots. However, the significance of this restriction is that we can markedly decrease the average number of time slots in a scheduling round. With a low average number of time slots we are safe to keep time slots in each scheduling round to a manageable number when letting short jobs run immediately on their arrivals without being queued.

To summarise, our strategy includes the following:

1. Backfilling technique is incorporated with gang scheduling, i.e., a limit is set for the maximum number of time slots in each scheduling round and the backfilling is applied to alleviate the blockade situation;
2. Jobs are divided into classes, for instance, short, medium and long and jobs of different classes are treated differently;
3. A limit (usually one) is set for large jobs to run simultaneously on the same processors to obtain an average low number of time slots in a scheduling round;
4. No special treatment is given to medium sized jobs;
5. The normal length of time slots is set reasonable long to minimise the scheduling overhead;
6. Let short jobs run immediately in the next time slot (instead of next scheduling round) to minimise their waiting time. If the required service time of a short job is shorter than the normal length of time slot, the time slot will be shortened as long as the short job can complete in a single time slot to reduce the average amount of idle time on each processor during the computation. (In our experiment the normal length of time slot may be set shorter than the service time of short jobs and then we will increase the length of the time slot for a short job such that the job can complete in a single time slot. In this way the performance of short jobs will not be affected greatly by changing the length of time slots.)

4 The System and Workload

The gang scheduling system for our experiment is mainly based on the job re-packing allocation strategy described in Section 2. In this experimental system, however, jobs can be classified into three classes, that is, short, medium and long, and in each experiment limits are set on how many time slots are allowed to be in a scheduling round and how many large jobs can run simultaneously on the same processors.

With the limit of time slots in a scheduling round introduced, we need to add a waiting queue to the system. If the limit is reached and a new job cannot be allocated to the existing time slots, it has to be queued. To alleviate the blockade problem the backfilling technique is adopted.

In our experiment the workload used is a synthetic workload generated from the parameters directly extracted from the actual ASCI Blue-Pacific workload [14]. We briefly describe this workload in the following paragraph and the detailed description can be found in [14, 15].

The workload is generated by using a modeling procedure proposed in [8]. It is assumed that parallel workload is often over-dispersive and then the job interarrival time distribution and job service time (or job length) distribution can be fitted adequately with Hyper Erlang Distribution of Common Order. The parameters used to generate a baseline workload are directly extracted from the actual ASCI Blue-Pacific workload. There are different workloads generated with different interarrival rates and average job length. In our experiment a set of 9 workloads is used. These workloads are generated by varying the model parameters so as to increase average job service time. For a fixed interarrival rate, increasing job service time increases the system workload. Each generated workload consists of 10,000 jobs and each job requires a set of processors varying from 1 to 256 processors.

In the next section we present some experimental results. We assume that there are 256 processors in the system. In each experiment we measure the average slowdown and the average number of time slots, which are defined as follows:

Assume the service time and the turnaround time for job i are t_i^e and t_i^r, respectively. The slowdown for job i is $s_i = t_i^r / t_i^e$. The average slowdown s is then defined as $s = \sum_{i=0}^{m} s_i/m$ for m being the total number of jobs.

If t_i is the time when there are i time slots in the system, the total computational time t_s will be $\sum_{i=0}^{l} t_i$ where l is the largest number of time slots encountered in the system during the computation. The average number of time slots in the system during the operation can then be defined as $n = \sum_{i=0}^{l} i t_i / t_s$.

5 Experimental Results

In our experiment we implemented three different strategies for resource allocation and reallocation for gang scheduling. They are named Strategy 1, 2 and 3 and briefly described as follows.

In Strategy 1 the job re-packing technique described in Section 2 is adopted and the limit of time slots in a scheduling round is set to 5. When the limit is reached, the incoming jobs have to be queued in a waiting queue. To alleviate the problem of blockade, the backfilling technique is applied. Note the limit of time slots is able to change in our experimental system and different limits will produce different simulation results. However, we are more interested in the relative performance, that is, the results of comparing different scheduling strategies. We have observed that this relative performance does not vary much with the change of the slot limit in a scheduling round. That is the reason we only show the result obtained by setting the slot limit to 5.

Strategy 2 is a simple extension of Strategy 1. The only difference is that the jobs are classified and long jobs are not allowed to time-share the same processors, that is, at any given time there is at most one long job running on each processor when Strategy 2 is applied.

Our new strategy is Strategy 3. It is the same as Strategy 2 except that a special treatment is also given to short jobs. When a short job arrives, it is immediately executed in the next time slot rather than the next scheduling round and the length of time slot is adjusted or varied such that the short job can complete in a single time slot.

In the following we present some results obtained from our experiment to compare these three strategies. The nine workloads are named ci for $0 \leq i \leq 8$ and the system workload becomes heavier when i increases. A job is considered short if its length is shorter than 150 and long if the job length is longer than 6000. With such setting, around 25% of the total jobs will be short jobs, another approximately 25% be long jobs, and the rest are considered as medium sized jobs.

We first compare Strategy 1 and Strategy 2. The experimental results for the comparison are obtained by setting the length of time slots to 1. As mentioned previously, the only difference between these two strategies is that long jobs are not allowed to run simultaneously on the same processors at any given time when Strategy 2 is adopted. We can see from Table 1 and Table 2 that by further limiting the number of long jobs on each processor both average slowdown for all jobs and average slowdown for short jobs are markedly decreased. This significant improvement in job performance is also seen when using the workloads generated from a different workload modeling procedure [17].

As shown in Table 2, when Strategy 2 is adopted, the average slowdown for short jobs decreases as the system workload becomes heavier. This is caused by the way the workloads are generated. As described in the previous section, all other 8 workloads (from c1 to c8) are generated from a baseline workload (c0) by increasing average job service time. In our experiment, however, we consider short jobs as those with a job length shorter than 150. When we measure average slowdown for short jobs, c1 will have less short jobs than c0 and c2 contains less short ones than c1, and so on. Therefore, we cannot claim that average slowdown for short jobs is decreased when the system workload becomes heavier.

Let t_i be the time when there are i time slots in the system and t_s be the total computational time. The values shown in Table 3 are the ratios of t_i and t_s for $1 \leq i \leq 5$ obtained by using different workloads. We can see from this table that most of the time (99% of the total operational time) the system is running with 5 time slots when Strategy 1 is adopted. When Strategy 2 is applied, however, we can see from Table 4 that a great amount of time the system is running with just 1 time slot (over 40% of the total time under heavy system workload). As a result, the average number of time slots in a scheduling round can be significantly reduced when Strategy 2 is adopted, as shown in Table 5.

Note that the average number of time slots decreases as the system workload becomes heavier for Strategy 2. This is because a job is considered long if its length is longer than 6000 in our experiment, and more long jobs will be produced due to the way the workloads are generated and thus more jobs could be queued when the workload becomes heavier.

The experimental results presented in the above tables are obtained by setting the length of time slot to 1. Table 6 and Table 7 show some results obtained by setting different slot length when Strategy 2 is applied. It can be seen from Table 6 that the overall average slowdown will increase as the length of time slot becomes longer. However, it is short jobs that suffer the most. As shown in Table 7, the average slowdown for short jobs can increase by a factor of more than 10 when the length is increased from 1 to 200.

It is not desirable to have over 20 minutes to finish the computation of a job having only 1 minute service time. Our third strategy is thus introduced to alleviate this undesirable situation. Our experimental results in Table 8 and Table 9 show that applying Strategy 3 can significantly enhance the performance of short jobs without greatly degrading the performance of others. The comparison of Table 8 with Table 6 shows that Strategy 3 performs a bit better in terms of the average slowdown for all jobs though not much. When comparing Table 9 with Table 7, however, we can see a big improvement for short jobs. Using Strategy 3 short jobs are not queued and are executed as soon as possible without considering if the limit of time slots in the system is exceeded. However, the average number of time slots in a scheduling round is not significantly increased, which can be seen by comparing the results obtain by setting the length of time slot to 200 and using Strategy 3 in Table 10 and those obtained by setting the length of time slot to 1 and using Strategy 2 in Tables 4 and 5.

6 Conclusions

It is known that space-sharing scheduling can cause the problem of blockade under heavy workload and that this problem can be alleviated by applying the gang scheduling strategy. Although there have been many new strategies introduced to improve the performance of gang scheduling and to make it more practical, the problem on how to set length of time slots has not been so far seriously addressed. This problem is very important and can seriously affect the performance of gang scheduling. In practice the length of time slot should be set relatively long as a short length of time slot can greatly increase the scheduling overhead. However, setting the length too long will degrade both efficiency of system utilisation and performance of job execution, especially short jobs.

In this paper we proposed a new strategy for alleviating the problem on how to set slot length. In this strategy short jobs are given special treatments, that is, short jobs will not be queued and are able to run as soon as possible on their arrival to minimise their waiting time, and the length of time slot may be temporarily adjusted as long as each short job can complete its execution in just a single time slot to minimise the average idle time on each processor. Therefore, the performance of short jobs will not be affected much by the length of time slot.

It should be stressed that limiting long jobs to run simultaneously on the same processors has played a very important role for the success of our new strategy. This is because to limit long jobs running simultaneously on the same processors can significantly reduce the average number of time slots in a scheduling round and thus there are more free places for short jobs to fit in without adding too many extra time slots to exceed the limit of a manageable number of time slots in the system.

In the paper we also presented some experimental results to demonstrate the effectiveness of our new strategy.

References

[1] A. Batat and D. G. Feitelson, Gang scheduling with memory considerations, *Proceedings of 14th International Parallel and Distributed Processing Symposium*, Cancun, May 2000, pp.109-114.

[2] A. B. Downey, A parallel workload model and its implications for processor allocation, *Proceedings of 6th International Symposium on High Performance Distributed omputing*, Aug 1997.

[3] A. C. Dusseau, R. H. Arpaci and D. E. Culler, Effective distributed scheduling of parallel workloads, *Proceedings of ACM SIGMETRICS'96 International Conference*, 1996.

[4] D. G. Feitelson, Packing schemes for gang scheduling, In *Job Scheduling Strategies for Parallel Processing*, D. G. Feitelson and L. Rudolph (eds.), Lecture Notes Computer Science, Vol. 1162, Springer-Verlag, 1996, pp.89-110.

[5] D. G. Feitelson and L. Rudolph, Distributed hierarchical control for parallel processing, *Computer* 23, 5, May 1990, pp.65-77.

[6] D. G. Feitelson and L. Rudolph, Job scheduling for parallel supercomputers, in Encyclopedia of Computer Science and Technology, Vol. 38, Marcel Dekker, Inc, New York, 1998.

[7] D. G. Feitelson and L. Rudolph, Gang scheduling performance benefits for fine-grained synchronisation, *Journal of Parallel and Distributed Computing*, 16(4), Dec. 1992, pp.306-318.

[8] J. Jann, P. Pattnaik, H. Franke, F. Wang, J. Skovira and J. Riordan, Modeling of workload in MPPs, *Proceedings of the 3rd Annual Workshop on Job Scheduling Strategies for Parallel Processing*, April 1997, pp.95-116.

[9] D. Lifka, The ANL/IBM SP scheduling system, In *Job Scheduling Strategies for Parallel Processing*, D. G. Feitelson and L. Rudolph (Eds.), Lecture Notes Computer Science, Vol. 949, Springer-Verlag, 1995, pp.295-303.

[10] J. K. Ousterhout, Scheduling techniques for concurrent systems, *Proceedings of Third International Conference on Distributed Computing Systems*, May 1982, pp.20-30.

[11] J. Skovira, W. Chan, H. Zhou and D. Lifka, The EASY - LoadLeveler API project, In *Job Scheduling Strategies for Parallel Processing*, D. G. Feitelson and L. Rudolph (Eds.), Lecture Notes Computer Science, Vol. 1162, Springer-Verlag, 1996.

[12] P. G. Sobalvarro and W. E. Weihl, Demand-based coscheduling of parallel jobs on multiprogrammed multiprocessors, In *Job Scheduling Strategies for Parallel Processing*, D. G. Feitelson and L. Rudolph (Eds.), Lecture Notes Computer Science, Vol. 949, Springer-Verlag, 1995.

[13] K. Suzaki, H. Tanuma, S. Hirano, Y. Ichisugi and M. Tukamoto, Time sharing systems that use a partitioning algorithm on mesh-connected parallel computers, *Proceedings of the Ninth International Conference on Distributed Computing Systems*, 1996, pp.268-275.

[14] Y. Zhang, H. Franke, J. E. Moreira and A. Sivasubramaniam, Improving parallel job scheduling by combining gang scheduling and backfilling techniques, *Proceedings of 14th International Parallel and Distributed Processing Symposium*, Cancun, May 2000, pp.133-142.

[15] Y. Zhang, H. Franke, J. E. Moreira and A. Sivasubramaniam, An integrated approach to parallel scheduling using gang-scheduling, backfilling and migration, *Proceedings of 7th Annual Workshop on Job Scheduling Strategies for Parallel Processing*, Boston, June 2001, pp.109-127.

[16] B. B. Zhou, R. P. Brent, C. W. Johnson and D. Walsh, Job re-packing for enhancing the performance of gang scheduling, *Proceedings of 5th Workshop on Job Scheduling Strategies for Parallel Processing*, San Juan, April 1999, pp.129-143.

[17] B. B. Zhou and R. P. Brent, Gang scheduling with a queue for large jobs, In *Proceedings of 15th International Parallel and Distributed Processing Symposium*, San Francisco, April 2001.

	c0	c1	c2	c3	c4	c5	c6	c7	c8
Strategy 1	29.6	89.2	156	179	252	307	368	357	405
Strategy 2	9.62	15.4	19.5	24.6	29.6	34.6	39.7	44.2	49.8

Table 1. Average slowdown for all jobs obtained by using Strategy 1 and Strategy 2.

	c0	c1	c2	c3	c4	c5	c6	c7	c8
Strategy 1	67.4	223	236	368	548	686	852	765	919
Strategy 2	4.02	4.34	3.68	3.42	2.76	2.61	2.52	2.41	2.37

Table 2. Average slowdown for short jobs obtained by using Strategy 1 and Strategy 2.

slots	c0	c1	c2	c3	c4	c5	c6	c7	c8
1	0.029	0.000	0.002	0.001	0.001	0.001	0.002	0.001	0.000
2	0.035	0.002	0.001	0.003	0.001	0.001	0.001	0.001	0.001
3	0.061	0.003	0.002	0.003	0.002	0.002	0.002	0.002	0.003
4	0.095	0.004	0.005	0.002	0.001	0.001	0.002	0.001	0.001
5	0.781	0.990	0.990	0.991	0.994	0.995	0.994	0.994	0.995

Table 3. The time ratio for different time slots obtained by using Strategy 1.

slots	c0	c1	c2	c3	c4	c5	c6	c7	c8
1	0.139	0.214	0.267	0.322	0.371	0.406	0.444	0.478	0.503
2	0.283	0.253	0.295	0.260	0.270	0.272	0.280	0.286	0.277
3	0.271	0.266	0.245	0.243	0.231	0.210	0.191	0.162	0.157
4	0.170	0.152	0.111	0.104	0.080	0.075	0.062	0.051	0.047
5	0.137	0.155	0.082	0.071	0.049	0.038	0.023	0.022	0.016

Table 4. The time ratio for different time slots obtained by using Strategy 2.

	c0	c1	c2	c3	c4	c5	c6	c7	c8
Strategy 1	4.57	4.98	4.98	4.98	4.99	4.99	4.99	4.99	4.99
Strategy 2	2.88	2.70	2.45	2.34	2.17	2.07	1.94	1.85	1.80

Table 5. The average number of time slots obtained by using Strategy1 and Strategy 2.

slot length	c0	c1	c2	c3	c4	c5	c6	c7	c8
1	9.62	15.4	19.5	24.6	29.6	34.6	39.7	44.2	49.8
10	10.0	14.7	20.5	25.2	29.6	35.3	39.5	44.5	49.8
50	11.7	16.7	21.1	27.5	31.3	36.7	40.6	46.3	50.1
100	14.1	19.4	23.4	26.9	34.2	38.2	44.5	49.1	52.3
150	17.4	22.4	26.2	30.9	35.7	41.1	45.4	49.9	55.3
200	20.2	24.9	29.1	34.8	39.0	43.6	47.6	53.0	56.3

Table 6. Average slowdown for all jobs obtained by varying the length of time slots, using Strategy 2.

slot length	c0	c1	c2	c3	c4	c5	c6	c7	c8
1	4.02	4.34	3.68	3.42	2.76	2.61	2.52	2.41	2.37
10	4.61	4.12	3.88	3.57	3.01	3.02	2.76	2.70	2.66
50	8.63	8.50	7.74	7.92	6.94	6.66	6.35	6.19	5.84
100	14.5	14.7	13.3	13.5	12.9	11.9	11.3	11.2	11.1
150	21.4	21.2	19.3	19.0	18.1	17.1	16.3	15.9	16.1
200	28.1	28.3	27.0	26.5	24.6	23.3	22.0	21.6	21.1

Table 7. Average slowdown for short jobs obtained by varying the length of time slots, using Strategy 2.

slot length	c0	c1	c2	c3	c4	c5	c6	c7	c8
1	8.40	13.4	19.1	23.9	28.9	34.9	40.3	43.5	48.8
10	8.66	13.7	19.8	24.8	29.9	34.2	39.7	44.2	49.0
50	9.11	14.0	19.6	24.3	30.1	35.2	40.7	44.4	49.0
100	8.97	14.6	19.8	25.0	30.8	34.2	41.1	44.8	50.0
150	9.93	15.7	20.7	25.4	30.9	36.5	40.6	46.3	50.6
200	11.3	16.5	21.7	26.1	31.2	37.6	42.1	47.8	52.0

Table 8. Average slowdown for all jobs obtained by varying the length of time slots, using Strategy 3.

slot length	c0	c1	c2	c3	c4	c5	c6	c7	c8
1	1.10	1.05	1.05	1.05	1.05	1.05	1.05	1.05	1.05
10	1.06	1.05	1.05	1.06	1.06	1.06	1.06	1.06	1.06
50	1.26	1.28	1.25	1.22	1.24	1.23	1.23	1.21	1.20
100	2.06	2.10	2.03	1.97	1.94	1.84	1.75	1.67	1.72
150	3.28	3.40	3.22	3.12	2.86	2.88	2.77	2.77	2.45
200	5.03	4.74	4.87	4.64	4.36	4.34	4.00	3.81	3.69

Table 9. Average slowdown for short jobs obtained by by varying the length of time slots, using Strategy 3.

slots	c0	c1	c2	c3	c4	c5	c6	c7	c8
1	0.154	0.216	0.275	0.320	0.372	0.409	0.445	0.481	0.510
2	0.245	0.239	0.251	0.244	0.234	0.251	0.253	0.244	0.262
3	0.263	0.249	0.247	0.240	0.234	0.215	0.198	0.183	0.156
4	0.175	0.152	0.116	0.122	0.101	0.088	0.069	0.066	0.051
5	0.163	0.143	0.111	0.075	0.058	0.037	0.036	0.026	0.021
6	0.001	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
average	2.95	2.77	2.54	2.39	2.24	2.09	2.00	1.91	1.81

Table 10. The time ratio and the average number of time slots (the last row) obtained by setting the slot length to 200 and using Strategy 3.