University of Denver Technical Report COMP-00-01
Improving Small Job Response Time for Opportunistic Scheduling *

Gaurav D. Ghare Scott T. Leutenegger
Mathematics and Computer Science Department
University of Denver Denver, CO 80208-0189
{ghare,leut}@cs.du.edu

Abstract

Opportunistic scheduling utilizes idle workstations to execute batch jobs. Other than ensuring that each batch user receives a fair allocation of service, no work addressing opportunistic scheduling of batch jobs has been done. In this paper we propose two simple scheduling policies to be used in conjunction with the Condor Up-Down scheduling algorithm. We show that an order of magnitude reduction in mean job slowdown can be achieved by rotating a user's batch jobs within their queue. Furthermore, this improved slowdown can be achieved without a significant loss of throughput. The reduced slowdowns result in a more interactive nature of the system thus increasing its appeal. Our proposed modifications to the algorithms would require very little effort to implement.

1 Introduction

Round robin and multilevel feedback queue scheduling are used in uniprocessor systems to provide interactive response times and good service for short jobs. These policies have not been implemented in opportunistic schedulers, probably because the primary objective of opportunistic schedulers has been maximizing throughput. The main contributions of this paper are a simulation based demonstration that round robin scheduling can be used in an opportunistic setting to improve both mean response times and mean job slowdowns at the expense of a small throughput reduction. Furthermore, a simple variation of FCFS, which we call the JR algorithm as described below, can provide much of the benefits of round robin with no reduction in throughput.

Opportunistic scheduling, also referred to as high throughput computing, has become an important way to maximize use of networks of workstations. By using idle workstations to execute batch jobs, opportunistic scheduling allows users to obtain significant CPU cycles that would not otherwise be available to them. The primary opportunistic scheduling system example is the Condor system developed at the University of Wisconsin [16, 18]. In Condor, users submit batch jobs to the batch system. A monitoring daemon is responsible for detecting machines that become idle. Once a machine is determined to be idle, it can be used to run batch jobs. An allocation daemon is responsible for selecting a job from the batch queue, migrating the job from the batch queue to the idle machine, and beginning execution of the batch job. If an idle workstation executing a batch job is reclaimed by the workstation owner, i.e.

*This work is supported by the NSF grant ACI-9733658.

new keyboard activity is detected or the load increases, the batch job is suspended, check-pointed, and returned to the batch queue. As shown in past studies [20, 21], and in current condor pool statistics [1], it is not unusual for a cluster of workstations to be more than 50% idle on average. Thus, by utilizing idle cycles, significant computation can be performed without interfering with workstation owner usage.

Opportunistic scheduling has been extended to work on geographically distributed systems [8, 11, 30], allow integration with multiple architectures [19, 27], support efficient checkpointing [17], support limited forms of parallelism [2, 4, 6, 7, 24, 25, 26], and minimize network demand [5]. Furthermore, opportunistic scheduling ideas have also been applied to harvesting idle remote memory in distributed memory systems [3, 9, 10, 12, 15, 23, 28, 29].

The primary goal of this past work has been to develop mechanisms to maximize throughput and add user functionality. Little work has been done on how best to schedule batch jobs. As shown from time-sharing systems, the scheduling algorithm can have a significant impact on user satisfaction. One of the most important issues, ensuring fair allocation of resource among different users submitting batch jobs, was addressed in an earlier paper [22]. This earlier paper proposed the "Up-Down" algorithm to ensure fairness among users. Each job is tagged with a user id. Total CPU allocation is kept track of per user. When jobs are removed from the batch queue for service, jobs belonging to the user with the least total CPU allocation are given priority. Within a user's jobs scheduling is FCFS. Other than the Up-Down algorithm, the authors know of no work specifically focusing on opportunistic scheduling, yet issues remain to be addressed.

The primary focus of this paper is to determine if the scheduling policy can reduce mean batch job slowdown, i.e. decrease the turnaround time of shorter batch jobs. A secondary question addressed is whether a policy can improve overall mean job response time. To illustrate the need for better turnaround time of short batch jobs consider the following scenario. A computational scientist runs thousands of numerical or discrete event simulations to explore the parameter space of the system under study. Often some jobs are of long duration while others are of short duration. By completing the shorter jobs quickly, the scientist gets feedback which may allow the scientist to either drop this parameter space region from the study or convince the scientist to focus on this region in detail. Thus, faster feedback will allow the scientist to replace some existing batch jobs with more relevant batch jobs, thus speeding up the discovery process. Another common scenario observed while the second author was in residence at NASA ICASE is as follows. A scientist runs long production run experiments in one portion of the parameter space. Meanwhile, the scientist explores other portions of the parameter space. These exploratory jobs are many, but shorter in length than the long jobs. The scientist does not want to terminate jobs that may have been running for days yet would like to get a set of shorter jobs to complete first. This can only be accomplished if these shorter exploratory jobs are not blocked by the previously submitted long production jobs.

To address these issues we propose two algorithms with which we augment the Up-Down algorithm. We propose that the Up-Down algorithm still be used to ensure fairness between users, but that our algorithms be used to better manage jobs within each user's queue of submitted jobs. Specifically we consider the current FCFS policy, a round-robin policy (RR), and a hybrid that we name job rotate (JR). The JR algorithm offers low overhead like FCFS but is better able to serve short jobs in preference to long jobs regardless of arrival order. We note that both policies proposed are easy to implement given existing opportunistic scheduling mechanisms. In fact, the JR algorithm would only require modifying a few lines of code and would incur no additional overhead.

We conduct discrete event simulation studies assuming three different workloads that we believe cover the majority of likely opportunistic workloads. Using our simulation models, we conduct extensive parametric studies to show how each assumption affects the relative performance of the policies.

The rest of this paper is organized as follows. In Section 2 we describe our scheduling algorithms in more detail. In Section 3 we describe our simulation methodology including workloads. In Section 4 we present our experimental results, and in Section 5 we conclude.

2 Algorithms Description

Our algorithms are implemented to work in conjunction with the Up-Down algorithm. When choosing a job from the batch queue the jobs belonging to the user with least total past CPU allocation are chosen first. To simplify our descriptions we assume that batch queue jobs are segregated into separate queues, one queue per user. Thus, we will refer to the user's batch queue. Our three policies differ in how jobs are chosen from and returned to the user's batch queue, and whether allocation/deallocation is determined only by workstation reclamation or by time slicing as in uniprocessor RR scheduling algorithms.

We consider three policies:

1. **FCFS**: In the FCFS algorithm new batch jobs are placed at the end of the users batch queue. When a job is chosen for service it is removed from the front of the users batch queue. Once placed on a workstation, the job runs until completion or until it is evicted by the workstation due to the workstation being reclaimed by the workstation owner. When a job is evicted from a workstation it is check-pointed and returned to the *front* of the users batch queue.

2. **JR**: The JR (job rotate) algorithm is identical to the FCFS algorithm except the location jobs are placed in the batch queue after eviction from a workstation. Upon eviction, a job is check-pointed but then returned to the *end* of the user's batch queue. As a result jobs are rotated. The frequency of rotation varies with the frequency of evictions. At the extreme case of no evictions, JR does not differ from FCFS. But, assuming some rate of evictions, JR will result in more fair service for short jobs as they do not have to wait for all the jobs ahead of them in the queue to be completed/scheduled before they can receive service.

3. **RR**: The RR algorithm is similar to a uniprocessor RR algorithm. Like JR, jobs are removed from the front of the queue and returned to the tail of the queue. In addition, we set a quantum size Q. Once a batch job acquires Q units of service on a workstation, we preempt the job, place it at the tail of the queue, and reschedule in its place the job at the head of the user's queue. This results in a more fair distribution of workstation cycles among the user's jobs regardless of eviction frequency. The policy incurs additional overhead relative to FCFS and RR. We include this overhead in our simulations.

Parameter	Description	Default Value
W	Number of workstations	30
BU	Number of batch users	1
CX_b	Coefficient of variation of batch jobs	3
BS	Batch Size	60
CS	Context Switch Overhead	0.1
Q	Quantum Size	5
U_o	Owner Utilization	0.3
CX_o	Coefficient of variation of owner jobs	3
D_o	Mean Demand of Owner Jobs	10
D_b	Mean Demand of Batch User Jobs	100
AD	Availability Delay	5

Table 1: Notation

3 Simulations

In this section we present the simulation methodology used and the various parameters considered. The parameters which we consider in the simulations are listed along with their default values in Table 1.

3.1 System Model

In this section we describe the overall system simulation model. The arrival and service distributions are described in more detail in the following section.

We used the Simpack [14, 13] simulation language and collected 10% confidence intervals at a 90% confidence level. We collect 20 batches of 2500 simulated job completions each.

We simulate a system of W workstations, where BU users are submitting batch jobs. Each workstation services a stream of workstation owner processes. When a workstation becomes idle, we wait for AD minutes before claiming the workstation. This is similar to what is done in the real condor system. If the owner claims the workstation during this AD minute period, then the workstation remains unavailable to the opportunistic scheduling system. If the workstation remains idle even after the AD minute wait period, then the workstation is available to the scheduler to run batch jobs.

We include overhead for allocation and deallocation of jobs. First consider the FCFS and JR scheduling policies. When a workstation becomes idle the first job from the selected user's batch queue is chosen and begins execution after CS units. The delay CS is non-productive time representing the cost to transfer and initiate the job. If a new workstation owner job starts before CS units have expired we assume the batch job is returned to the batch queue at the same time as the start of the new workstation owner job. After beginning execution, a batch job runs until completion or eviction. If the batch job is evicted, time from start until eviction is decremented from the remaining service time of the the batch job. In a real system check pointing intervals may be used and the job is rolled back to the most recent checkpoint. This simplification shall not change the qualitative results presented in this paper. When a job is evicted, if there is another idle workstation the job is allocated to the idle workstation and begins execution at the eviction time plus CS units to represent the migration cost. If no other workstations are idle the job is returned to the batch queue and is available to contend for scheduling after the checkpointing has been completed (CS units later).

The RR algorithm differs in that once a batch job is scheduled it is preempted after Q units. At preemption time a delay of CS units is assumed to deschedule the job and return to the batch queue. Then an additional delay of CS occurs to schedule the next job. Thus, a complete scheduling cycle is of length $Q + 2 * CS$, of which $2 * CS$ is unproductive overhead.

3.2 Workload Model

We assume workstation owner jobs have an exponential inter-arrival time, and that the owner job service demand distribution is a two-stage hyper-exponential with coefficient of variation CX_o and mean D_o. Parameters CX_o and D_o are simulation inputs. The mean owner utilization, U_o, is the percentage of time the workstations are used by owner jobs and is a simulation input. Given D_o and U_o, the mean inter-arrival time is calculated.

Characterizing the batch workloads is difficult. One approach would be to measure a production system and derive a model based on measured values. Unfortunately the results would only be relevant to that particular system. One previous more general workload model was proposed in [21]. We wished to explore a larger parameter space and hence chose to consider three different workloads. The first, independent, is based on similar past models in the queuing theory literature. The second model, Batch, is based on the second author's observations while using the Condor system. Users tend to submit jobs in batches whenever the batch queue length is short and refrain from submission when the batch queue length is long. We have modeled the Batch workload to reflect this behavior. The third model, Two Class Normal, is based on the second authors experience in a computational science center where scientists run very long jobs (simulations for example), but may also need to run groups of short jobs concurrently (usually to help the user choose the next set of simulations). This results in batches of long jobs with frequent batches of short jobs being submitted while the longer jobs are still running. Thus, we consider the following workloads. Unless being varied, all parameters are assigned the default values as shown in Table 1.

1. **Batch:** In the Batch Arrivals workload, a user submits jobs in batches of size BS. Each batch user submits a new batch of jobs whenever the length of that user's batch job queue drops below BS. All jobs in a batch arrive at the same time. After submitting a batch the user does not submit jobs again until the queue length drops below BS again. The job demands are derived from a two-stage hyper-exponential distribution with each sample independent from the other jobs within the batch.
2. **Independent:** Batch jobs arrive independently. We assume an exponential inter-arrival time for the stream of jobs. We choose the mean inter-arrival time such that the system utilization, including both batch jobs and workstation owner jobs, is 90%. The job demands are derived from a two-stage hyper-exponential distribution.
3. **Two Class Normal:** The Two-Class Batch Normal workload assumes a system in which the user's

jobs belong to one of two classes: long or short. The mean demand of long jobs is 24 times the mean demand of the short jobs. The demands for the long and short jobs are derived from normal distributions with a standard deviation 1. A batch of long jobs is assumed to arrive whenever the user's job queue length drops below BS.

Parameter P_s, which is the ratio of submitted jobs that are short to the total number of jobs submitted, determines the mean inter-arrival time of the batches of short jobs. The inter-arrival times of batches of short jobs are chosen from an exponential distribution.

3.3 Metrics

We define job response time to be the difference between completion and submission times. The reported mean job response times are the mean over all completed simulation jobs.

We define job slowdown as the response time divided by the service demand. The mean slowdown reported in our figures is the mean over all completed simulation jobs. The slowdown metric emphasizes the quality of service delivered to the short jobs, and hence is a good measure of how interactive the system is. Since our focus is on reducing turnaround time of short jobs we focus on slowdown as our primary metric, but also include mean job response time.

4 Experimental Results

In this section we present our simulation results.

Since the up-down algorithm takes care of distributing the workstation cycles fairly among various users of the system, and the policies we consider vary only in the way in which each user's jobs are scheduled independent of the other users in the system, we assume only a single user for most of our simulations. We do study the effect of adding more users to the system on the performance of the scheduling policies, and show that the relative performance of the policies studied is the same for a multi-user system as for a single-user system.

All response time plots are in terms of unspecified units. If we assume one unit equals 1-5 minutes, then the default context switch overhead is 6-30 seconds, the default batch job mean demand length is 100-500 minutes, and the RR quantum size is 5-25 minutes. These are plausible numbers.

We first present results for the Batch Arrival Workload. Following this we present select results for the Independent Workload. Lastly we present results from the Two Class Normal workload. Rather than concoct experiments to attempt to make one policy look better than the others we conduct a sensitivity study of all parameters. In the process, the relative merit of the different algorithms becomes clear.

4.1 Batch Arrivals

We first consider the effect of maximum batch size, BS, on the relative performance of the policies. The effect of BS on FCFS, JR and RR is shown in Figure 1. As BS increases so do the mean response times and slowdowns of the policies. There is very little difference in the mean response times, but the slowdowns of FCFS and JR are considerably larger than the slowdown of RR. The poor performance of FCFS for larger BS can be attributed to the longer queue lengths and hence the wait times for short jobs is much greater than under JR and RR. The RR and JR and policies are better able to accommodate the larger batch sizes than FCFS since RR ensures, and JR attempts, to service the shorter jobs quickly by rotation. Note the slowdowns of FCFS and JR are about 2 and 9 times larger respectively than the corresponding slowdowns of RR.

In Figure 2 we plot mean response times and slowdowns versus owner utilization. We simulated owner utilization by keeping D_o constant while varying the frequency of arrival of owner jobs proportional to U_o. As the owner utilization increases, the idle time of the workstations and hence the time available for running batch jobs decreases. The decrease in available time, causes the mean response time of all the policies to increase. Again there is little difference in mean response times . The slowdowns differ considerably. FCFS performs much worse than RR and JR. At high values of U_o JR tends to perform similar to RR. This is because there are more frequent owner job arrivals, and thus the jobs tend to get rotated more frequently, causing a RR-like behavior. Note that when $U_o = 0$, JR is identical to FCFS, and at $U_o = 90\%$ JR performs identical to RR.

We next consider the impact of batch job length. In Figure 3 we plot the mean response times and slowdowns versus D_b. The mean response times are not affected, whereas the slowdowns are. The slowdowns of FCFS are considerably larger than those of RR which are larger than those of JR. An increase in D_b increases the slowdown of FCFS but decreases the slowdown of JR and RR. This can be explained as follows. When we increase D_b, there are more long jobs, hence the increase in FCFS slowdown. The slowdowns of JR and RR improve since an increase in job demand effectively decreases quantum size relative to job demand. Hence, under JR and RR all jobs are rotated more times as demand increases, each job receives fairer treatment, and thus the slowdown is reduced.

Next consider the effect of varying owner job length. In Figure 4 we plot mean response time and slowdown versus D_o. The policy mean response times do not differ significantly. However the slowdowns of FCFS are much worse than those of RR and JR. The slowdowns of FCFS and RR and relatively unaffected by D_o. The slowdowns of JR degrade upon increasing D_o. We maintain a constant owner utilization while varying D_o, hence as D_o is increased, we have fewer owner jobs, and the average length of each owner job increases. As the number of owner jobs decreases, the frequency of batch job rotation (in JR) also decreases, thus resulting in an increase of JR slowdown. When D_o is very small, JR tends to perform like RR and when D_o is very large, JR tends to perform like FCFS.

We next consider the effect of batch job demand variation. In Figure 5 we plot the mean response time and slowdown of the policies versus CX_b. The number of short jobs increases along with CX_b. Since short jobs are serviced better by RR and JR, the mean response time of RR and JR both decrease, whereas the mean response time of FCFS remains fairly constant. However the slowdowns of FCFS increase with CX_b as there are more short jobs, and their wait times relative to their demands are greater. The slowdowns of RR and JR also increase slightly, but are not as significantly affected.

We also varied the coefficient of variation of workstation owner jobs from 1 to 15, but it had little affect on mean response time or slowdown, and thus we omit the graphs from the paper.

We next consider the impact of context switch overhead. In Figure 6 we plot the mean response times and slowdowns for various values of CS. As CS increases, the mean response time of RR increases faster than the other polices since it performs significantly more context switches. The slowdowns of the policies are not as strongly affected by an increase in CS. In fact, despite RR having a higher mean response time as CS is increased, it still maintains much smaller slowdowns than FCFS and JR. The FCFS (JR) policy has slowdowns that are about 4 (2) times higher than RR. Thus, RR is better able to provide good turnaround times of short jobs even in the face of higher overheads.

We next consider the impact of quantum size. We plot the effect of mean response time and slowdown versus Q in Figures 7. For very small values of Q, RR spends a lot of time context switching, hence resulting in larger response times. As shown in the graphs, quantum size has little impact on performance other than the increase in RR mean response time at small values of Q.

We also varied the number of workstations available to the batch user. The results are shown in Figure 8. When the number of available workstations increases, all the policies show a significant improvement in performance. As W is increased, the relative difference in the policies reduces. Note that while varying W we kept the owner and batch utilization constant. The reason for the improvement in all the policies (when W is increased) is that it is easier for the short jobs to get through when there are more available workstations. In addition, as W increases, the frequency of owner job arrival in the system as a whole increases. This causes more frequent job rotations in JR. Thus JR tends to perform more like RR.

Finally, we considered the effect of varying the number of batch users. In Figure 9 we plot mean response time and slowdowns versus the number of batch users. When more users are added less time is available to each user, batch jobs experience longer wait times, and hence mean response times and slowdowns increase. The relative performance of RR and JR is not affected, but FCFS is significantly worse as the number of users is increased. Thus, by presenting results for one user in all of our other experiments we provide a lower bound on the difference between FCFS versus RR and JR.

4.2 Effect on Throughput

In this section we discuss the impact on throughput assuming the batch arrivals workload. In the left hand side of Figure 10 we plot the throughput of opportunistic jobs versus context switch overhead. At a context switch overhead value of 0.1 units, which corresponds to values of 6 to 30 seconds as discussed at the beginning of this section, RR scheduling results in 1% less throughput than FCFS. At extremely high context switch of 2 units, i.e. 2 to 10 minutes, there is a throughput loss of 20%. Initial measurements indicate Condor context switches can be performed in 15 seconds or less, thus a context switch parameter of 0.1 units is appropriate.

In the right hand side plot of Figure 10 we plot the throughput versus quantum size with context switch overhead, CS, set to 2 units and all other parameters set at the default value. Even at this unrealistically high value of CS, the decrease in throughput resulting from RR scheduling can be mitigated by increasing the quantum size. At a quantum size of 25 units, there is no significant difference in throughput. Furthermore, as shown in Figure 7, this larger value of Q does not significantly reduce the benefit of RR scheduling.

For all other previous experiments, which assume a context switch overhead of 0.1 units, the throughput difference between the three algorithms is 1% or less. Thus, the throughput reduction is not a significant concern.

In summary, based on initial real system context switch measurements, and the ability to mitigate context switch costs by varying quantum length, we expect RR scheduling will not have a significant impact on throughput. In the unlikely situation that context switch costs should prove to significantly impact throughput, then the JR algorithm should be used. The JR algorithm achieves much of the slowdown reduction benefit of RR, yet incurs no throughput reduction cost relative to FCFS.

4.3 Independent Arrivals

In this section we include results from our experiments assuming the independent arrivals workloads. In general the qualitative results are similar to the batch workload. We ran the same experiments as above for the batch workload. Only two experiments result in significant differences.

For the independent arrival the RR policy is much more sensitive to the context switch overhead. We plot the results in Figure 11.

The independent arrival workload also reveals a greater difference in mean response time and slowdown for smaller numbers of workstations than does the batch workload. We plot the results in Figure 12.

4.4 Two-Class Batch Normal

In this section we discuss the results for two class batch normal workload. We present the overall mean response times and slowdowns in Figure 13, the mean response times and slowdowns of the long job class in Figure 14, and the mean response times and slowdowns of the short job class in Figure 15. In all plots we plot the mean response time or slowdown versus the percent of the workload belonging to the short job class, P_s. As we can see from the plots, FCFS gives very poor slowdowns for the short jobs and the overall workload, but much better slowdowns for the long jobs. Thus, we can improve mean job response time and slowdown for long jobs by using FCFS, but at the expense of the short jobs which are the jobs we desire to provide rapid turnaround time. When we consider jobs belonging to the short class, JR and RR result in smaller mean response times and slowdowns than FCFS, but for practical purposes are the same.

5 Conclusion and Future Work

In this paper we have proposed using the JR or RR scheduling policies instead of FCFS to augment the Condor Up-Down algorithm. By conducting an extensive set of simulation studies, we have demonstrated that the JR and RR policies often result in slightly lower mean response times than FCFS and almost always result in significantly lower mean job slowdowns. Based on our simulation studies, on average the mean job slowdown of FCFS is about 8 times larger than that of RR, and the mean job slowdown of JR is about 2 times larger than that of RR. Note, the factor of 8 relative to FCFS should be viewed as a lower bound since as the number of batch users increases the ratio of FCFS slowdown to RR or JR slowdown increases. Furthermore, based on our initial experimental context switch cost measurements, these slowdown reductions cause only a 1% or less reduction is throughput.

By reducing the slowdown, i.e. reducing the turnaround time of the shorter jobs, opportunistic systems become more interactive than the regular batch system. This has two positive aspects. First, it increases the range of job demands for which opportunistic computing is acceptable. Second, instead of expecting results in days or weeks, the computational scientist can expect many results within hours or a day. This faster feedback may allow the scientist to cancel currently queued jobs and replace them with more productive jobs, thus decreasing total offered load.

Based on our simulations, it is clear that RR is preferable to JR despite the larger number of context switches necessary. This claim is obviously dependent on the actual overhead incurred by an RR context switch. In the event that context switches should be more costly than our initial measurements indicate, the JR algorithm, which incurs no overhead relative to FCFS, can be used to still greatly reduce slowdown relative to FCFS without any decrease in throughput. Thus, for minimal implementation effort an order of magnitude or more improvement in job slowdown can be achieved by using RR or JR scheduling.

We have recently begun implementing the RR and JR policies on Condor, using Condor provided scheduling support API level calls. Based on measured RR context switch overheads, we plan to tune quantum size Q to find the optimal tradeoff between overhead, hence throughput loss, and slowdown time reduction.

Acknowledgments. We would like to thank Miron Livny for useful discussions that influenced this work.

References

[1] The condor project. University of Wisconsin, URL: http://www.cs.wisc.edu/condor/.

[2] A. Acharya, G. Edjlali, and J. Saltz. The utility of exploiting idle workstations for parallel computation. In *Proc. 1997 ACM SIGMETRICS*, pages 225–236, 1997.

[3] A. Acharya and S. Setia. Availability and utility of idle memory in workstation clusters. In *Proc. of ACM SIGMETRCS 1999*, volume 27, pages 35–46, June 1999.

[4] R.H. Arpaci, A.C. Dusseau, A.M. Vahdat, L.T. Liu, T.E. Anderson, and D.A. Patterson. The interaction of parallel and sequential workloads on a network of workstations. In *Proc. 1995 ACM SIGMETRICS*, 1995.

[5] J. Basney and M. Livny. Improving goodput by co-scheduling cpu and network capacity. *International Journal of High Performance Computing Applications*, 13(3), 1999.

[6] N. Carriero, E. Freeman, D. Gelernter, and D. Kaminsky. Adaptive parallelism and piranha. *IEEE Computer*, pages 40–49, Jan 1995.

[7] K. Chanchio and X.-H. Sun. Mppvm: A software system for non-dedicated heterogeneous computing. In *Proc. of the International Conf. on Parallel Processing*, 1996.

[8] C. Chen, K. Salem, and M. Livny. The dbc: Processing scientific data over the internet. In *Proc. of 16th International Conference on Distributed Computing Systems (ICDCS 96)*, May 1996.

[9] D. Comer and J. Griffioen. A new design for distributed systems: The remote memory model. In *Proc. of the 1990 USENIX Summer Conference*, 1990.

[10] M. Dahlin, R. Wang, T. Anderson, and D. Patterson. Cooperative caching: Using remote memory to improve file system performance. In *Proc. of OSDI'94*, pages 267–80, November 1994.

[11] D.H.J. Epema, M. Livny, R. Van Dantzig, X. Evers, and J. Pruyne. A worldwide flock of condors : Load sharing among workstation clusters. *Journal on Future Generations of Computer Systems*, 12, 1996.

[12] M. Feeley, W. Morgan, F. Pighin, A. Karlin, H.Levy, and C. Thekkath. Implementing global memory management in a workstation cluster. In *Proc. of SOSP'95*, pages 201–12, December 1995.

[13] P. A. Fishwick. *Simulation Model Design and Execution: Building Digital Worlds.* Prentice Hall, 1995.

[14] P.A. Fishwick. Simpack: Getting started with simulation programming in c and c++. In *Proc. 1992 Winter Simulation Conference*, pages 154–162, 1992.

[15] L. Iftode, K. Li, and K. Petersen. Memory servers for multicomputers. In *COMPCON Spring'93 Digest of Papers*, pages 538–47, February 1993.

[16] M. Litzkow and M. Livny. Experience with the condor distributed batch system. In *Proc. of IEEE Workshop on Experimental Distributed Systems*, Oct 1990.

[17] M. Litzkow and M. Solomon. Supporting checkpointing and process migration outside the unix kernel. In *Usenix Conference Proceedings*, pages 283–290, January 1992.

[18] M. J. Litzkow, M. Livny, and M. W. Mutka. Condor - a hunter of idle workstations. In *Proc. of the 8th International Conference on Distributed Computing Systems (ICDCS)*, pages 104–111, 1987.

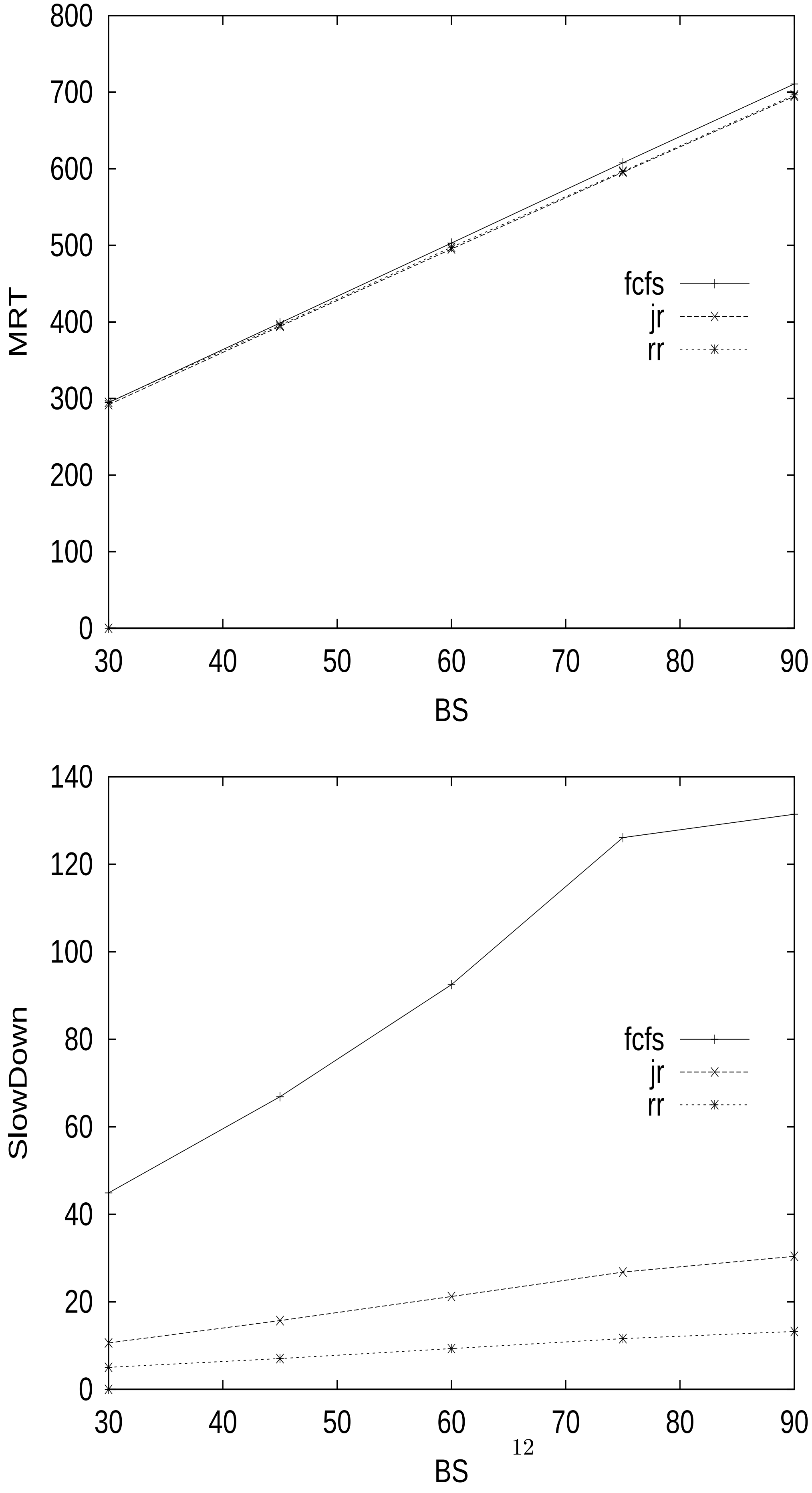

Figure 1: Effect of Batch Size, BS

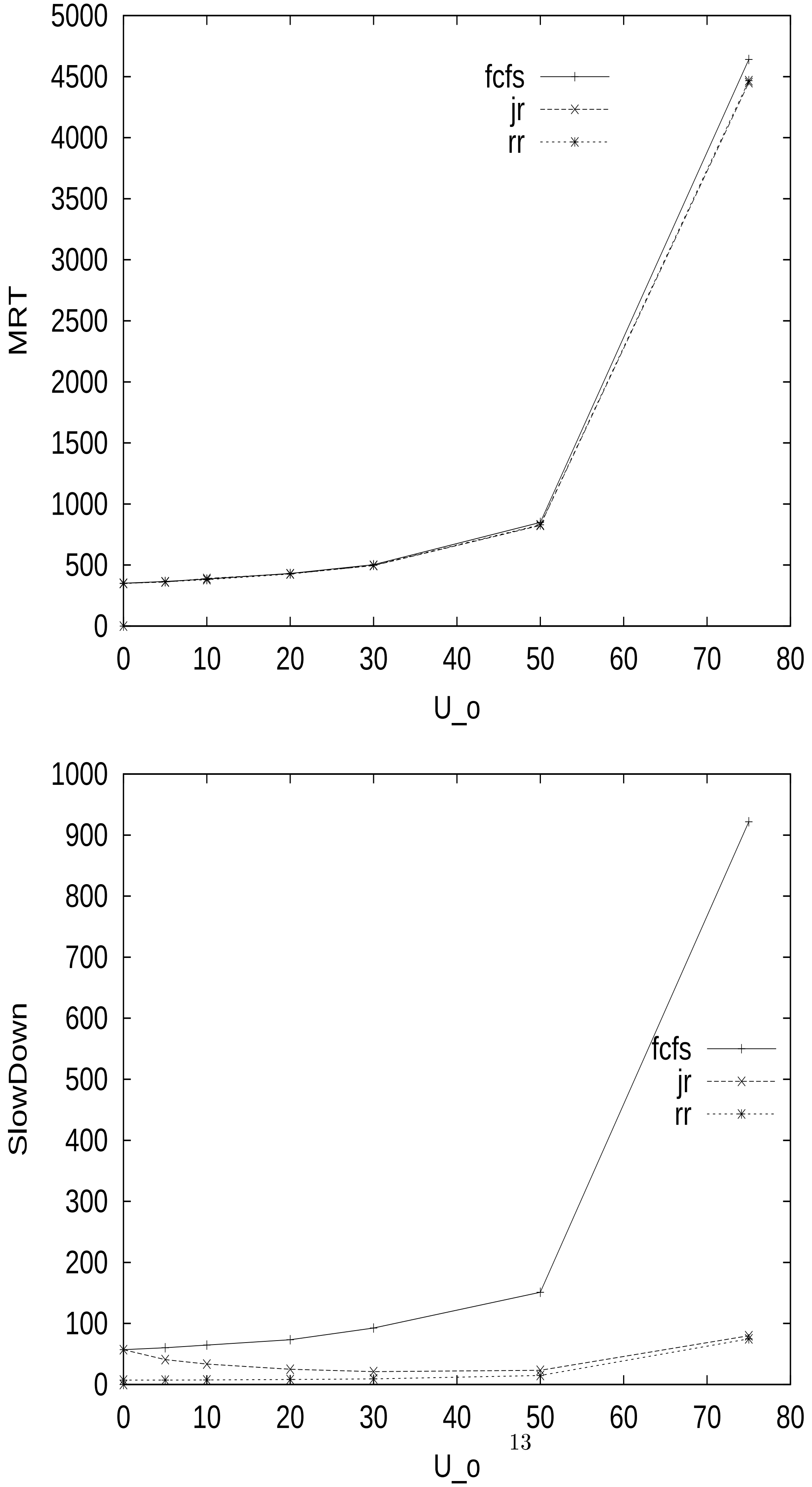

Figure 2: Effect of Owner Utilization, U_o

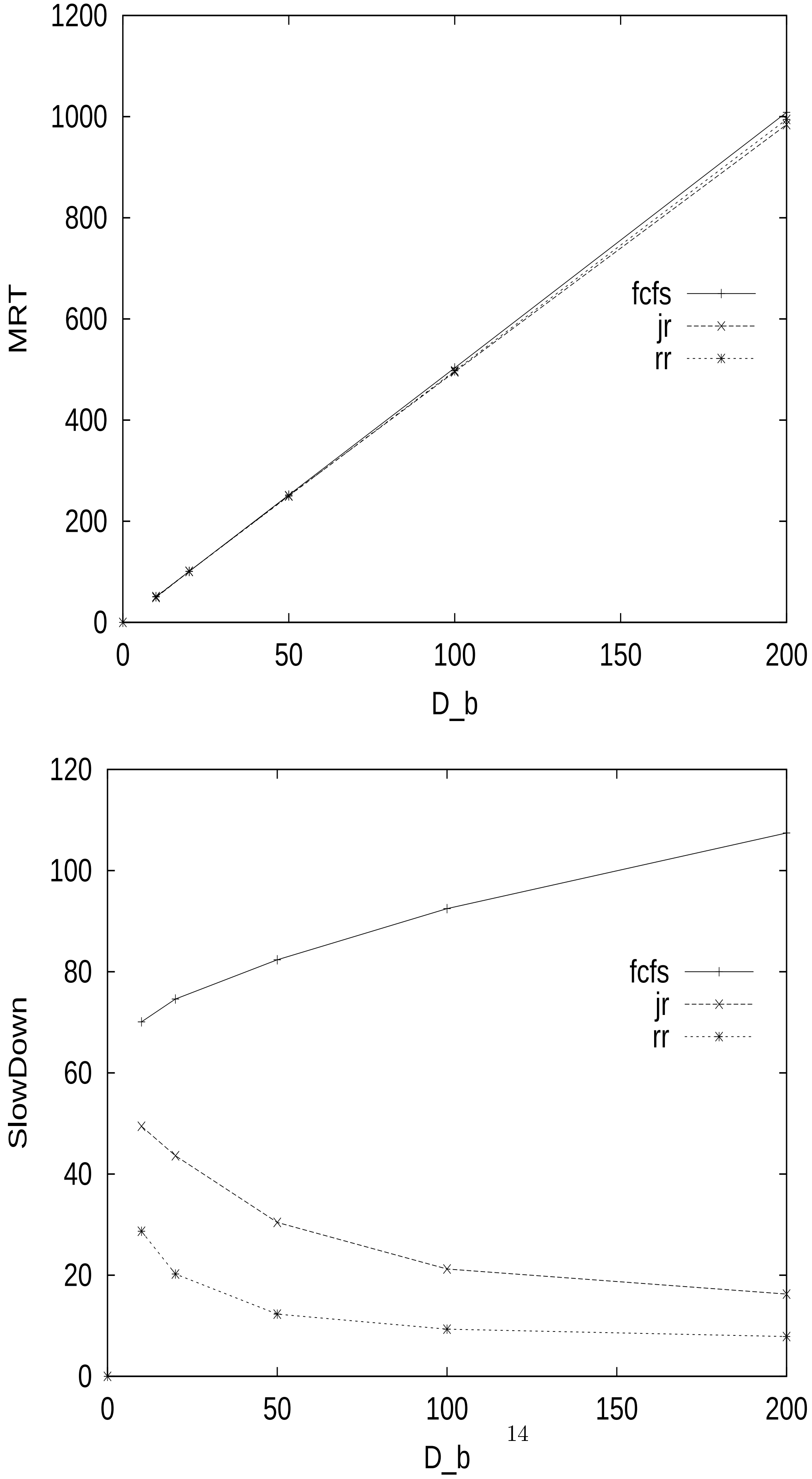

Figure 3: Effect of mean batch job demand

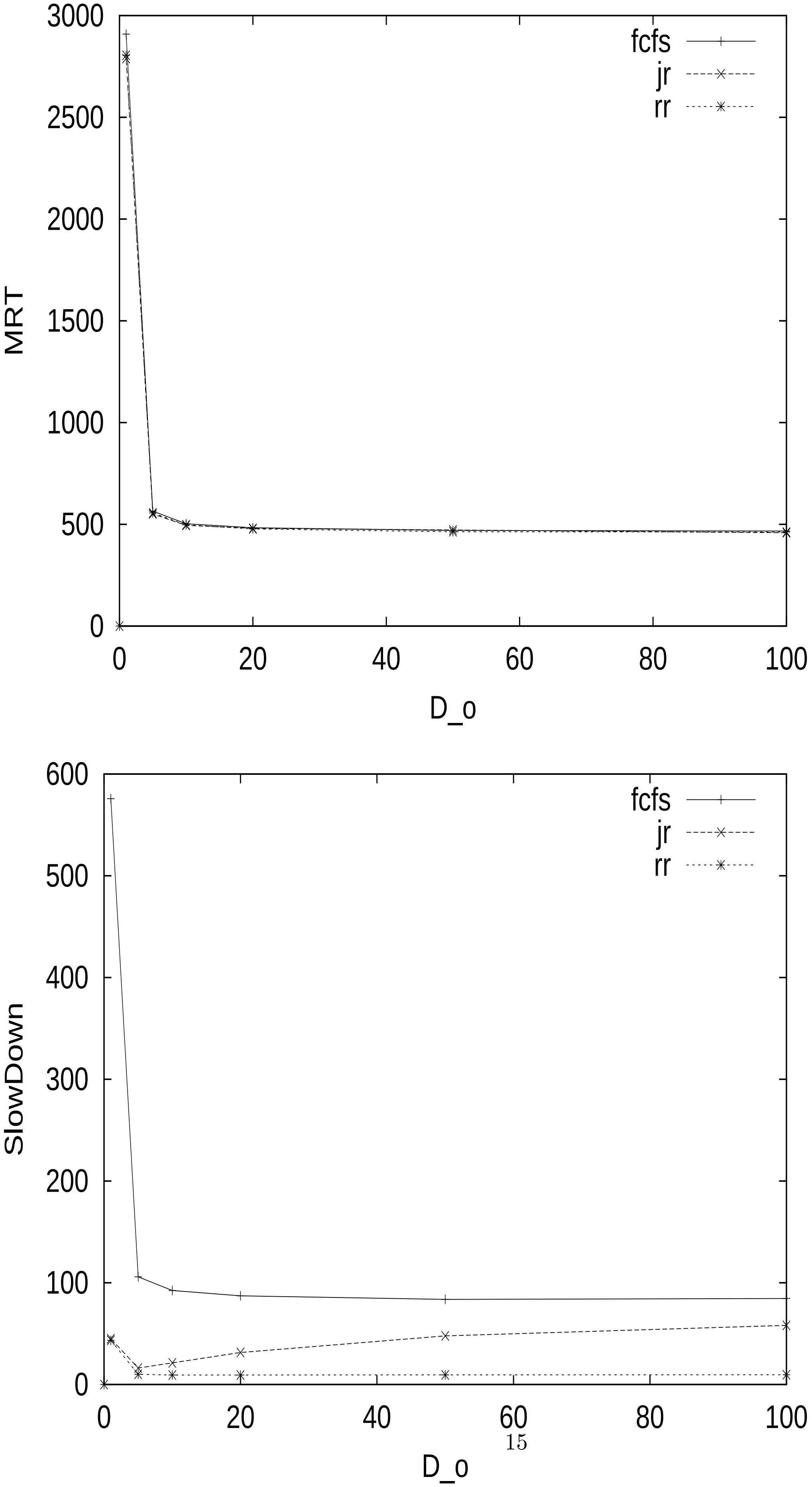

Figure 4: Effect of mean owner job demand

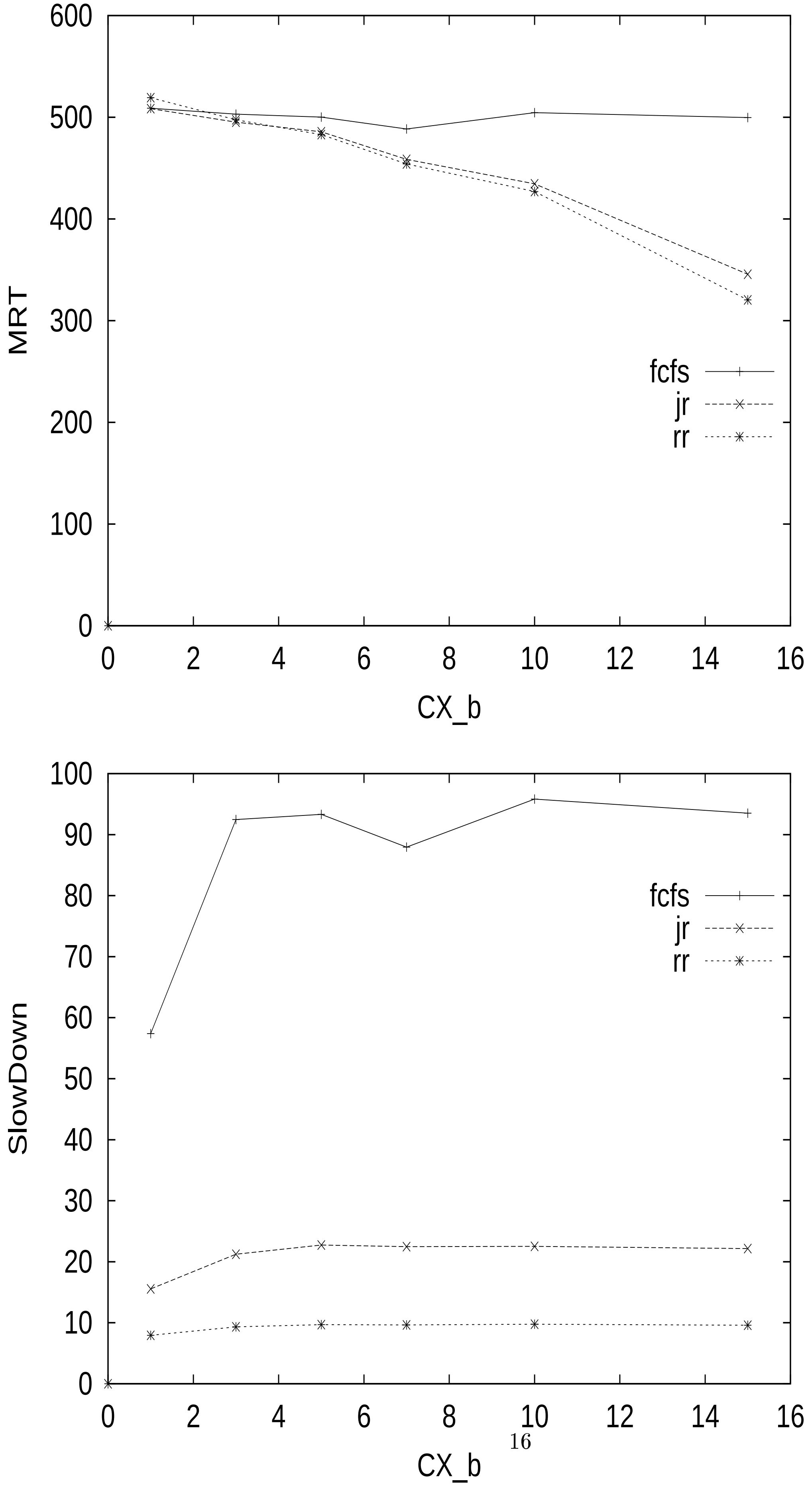

Figure 5: Effect of batch job demand variation

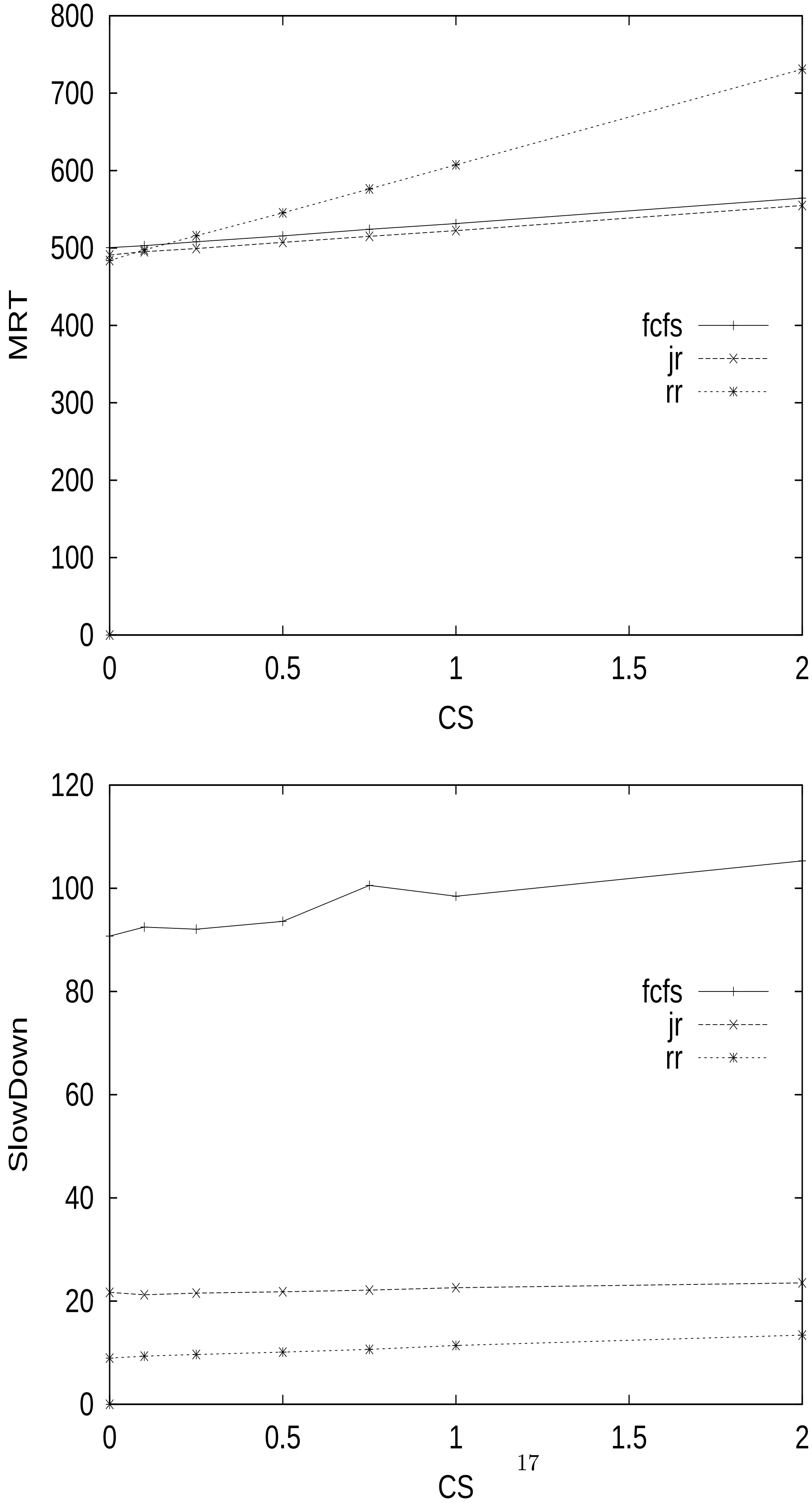

Figure 6: Effect of context switch overhead

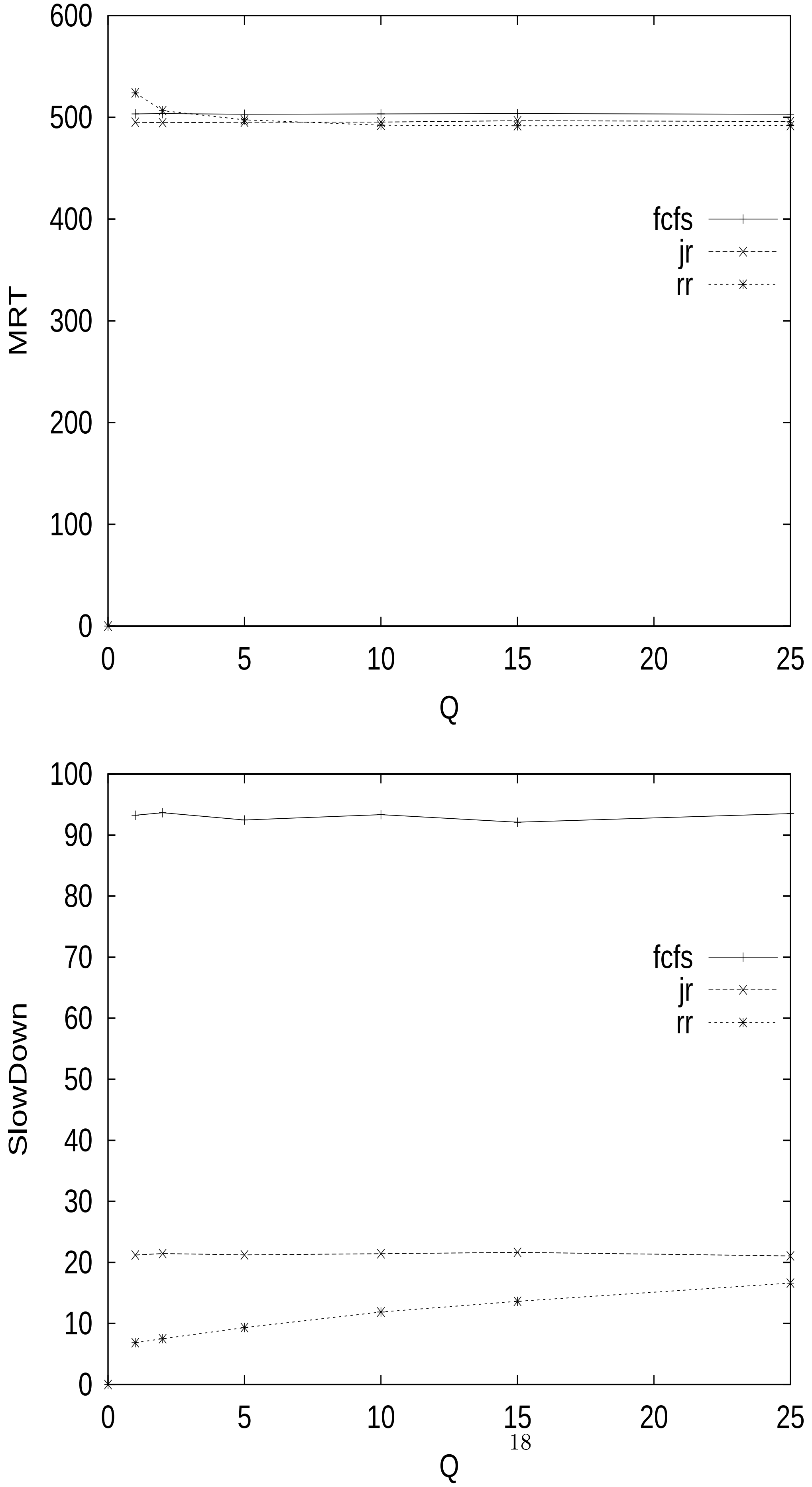

Figure 7: Effect of quantum size

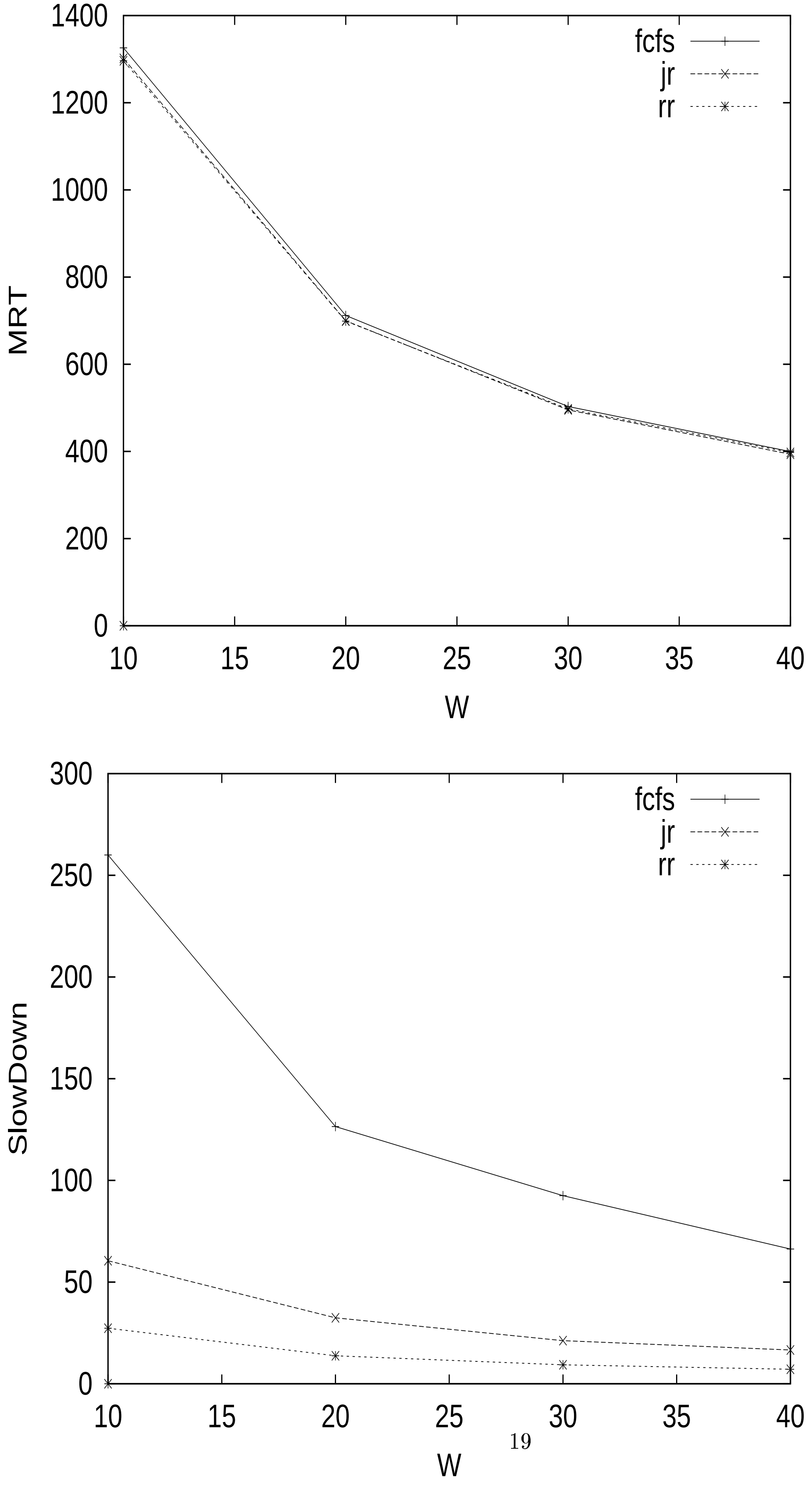

Figure 8: Effect of number of workstations

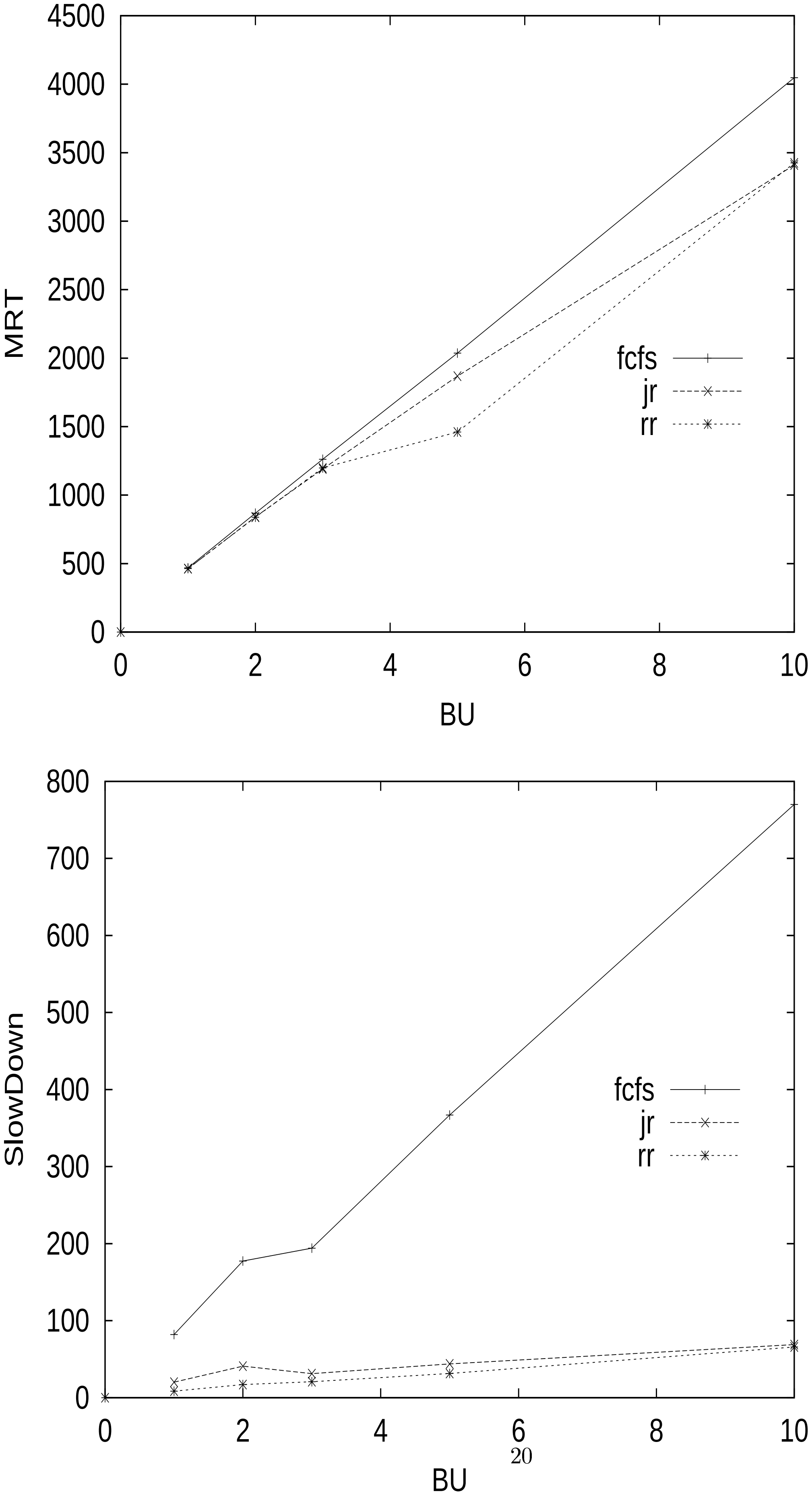

Figure 9: Effect of number of batch users

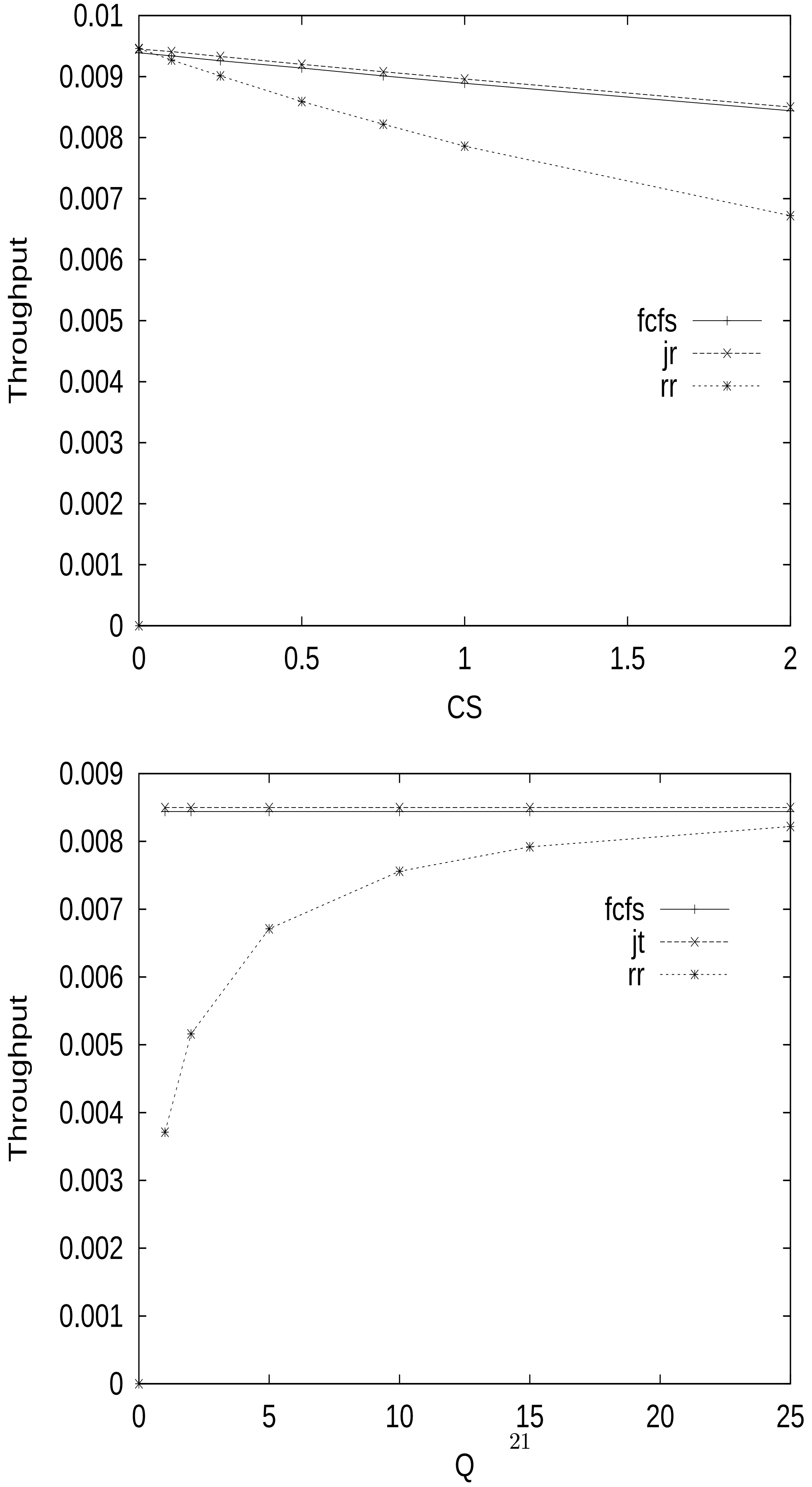

Figure 10: Effect of context switch and quantum size on Throughput

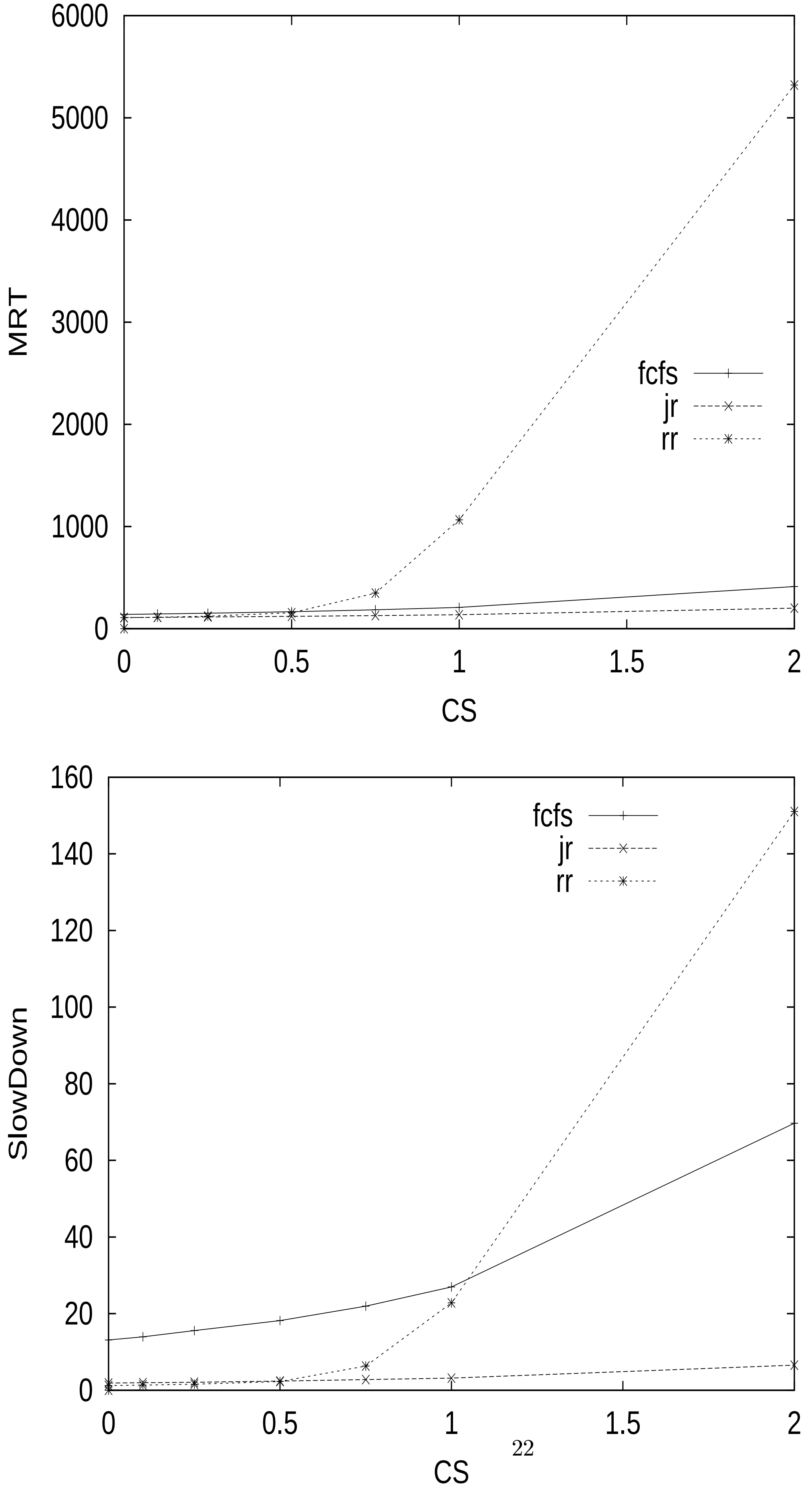

Figure 11: Independent Arrivals: Effect of context switch overhead

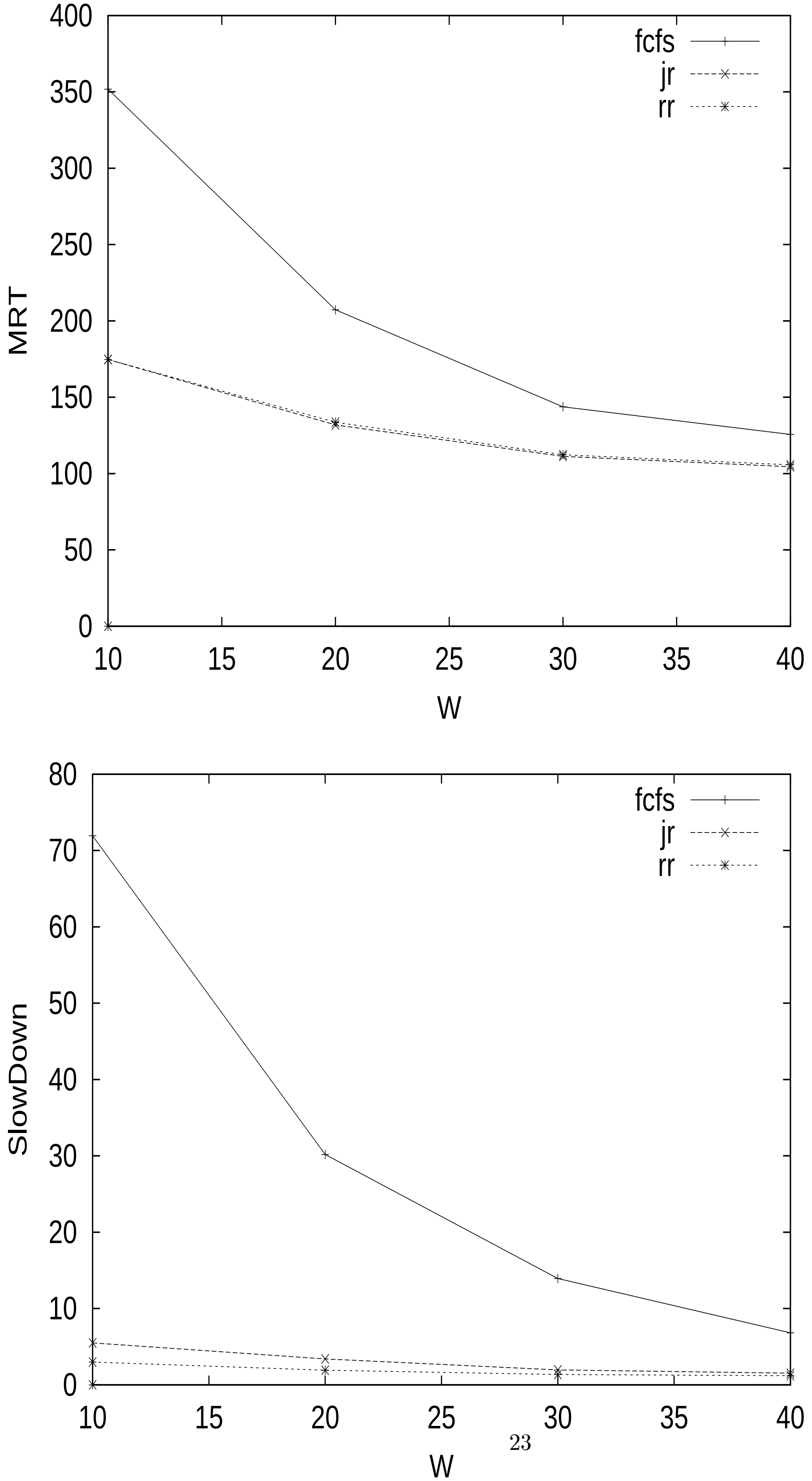

Figure 12: Independent Arrivals: Effect of number of workstations

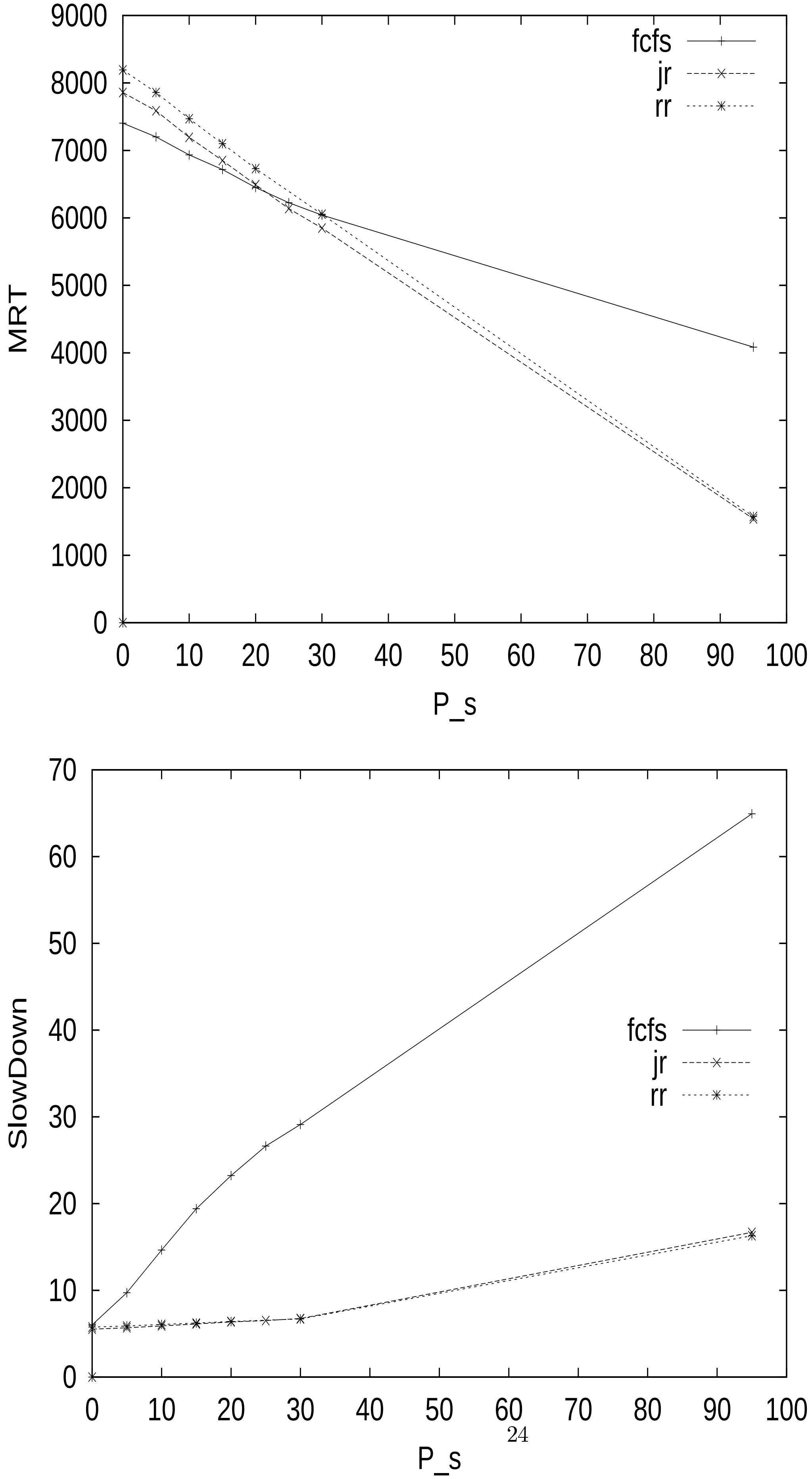

Figure 13: Two Class Normal

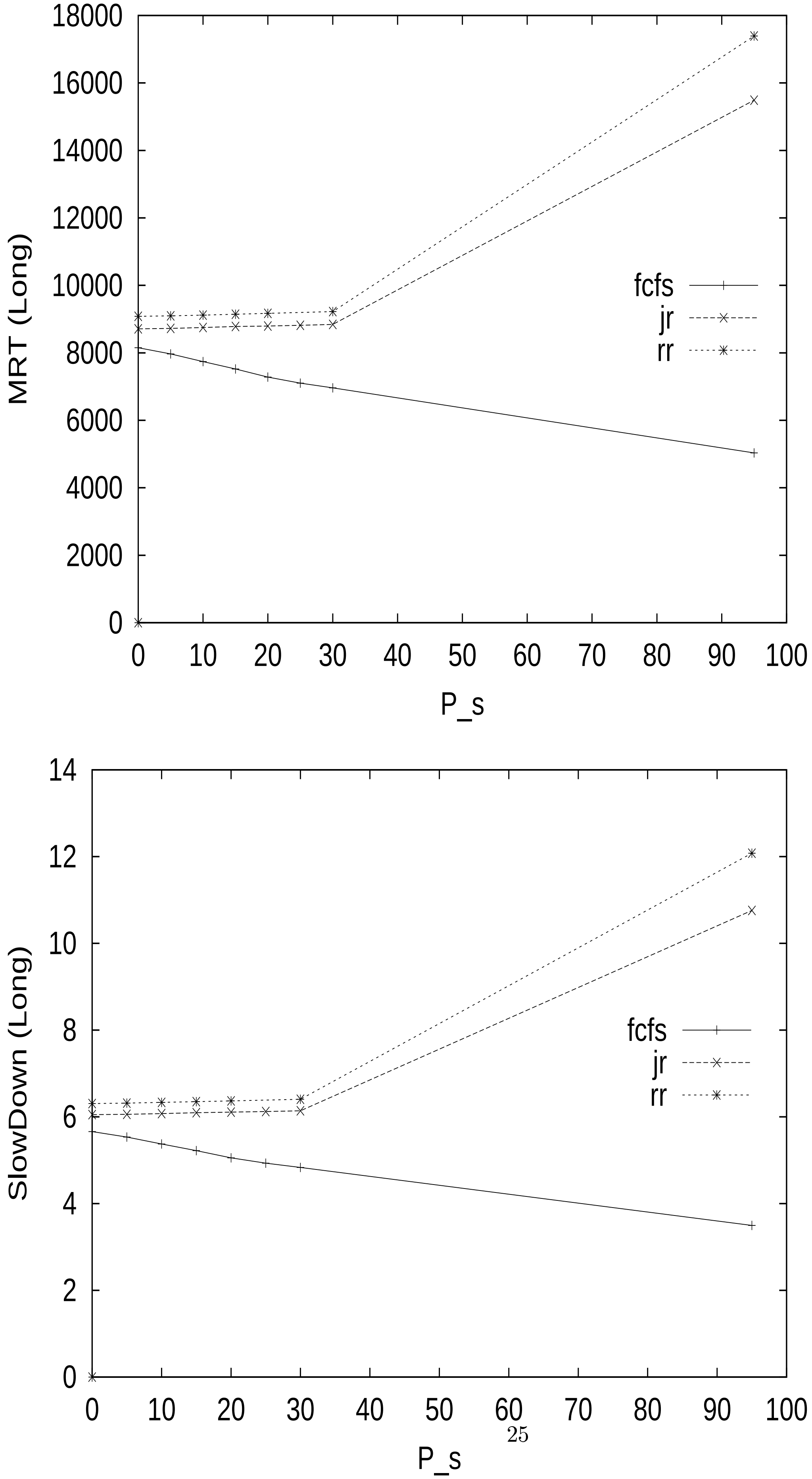

Figure 14: Two Class Normal: Long Jobs Only

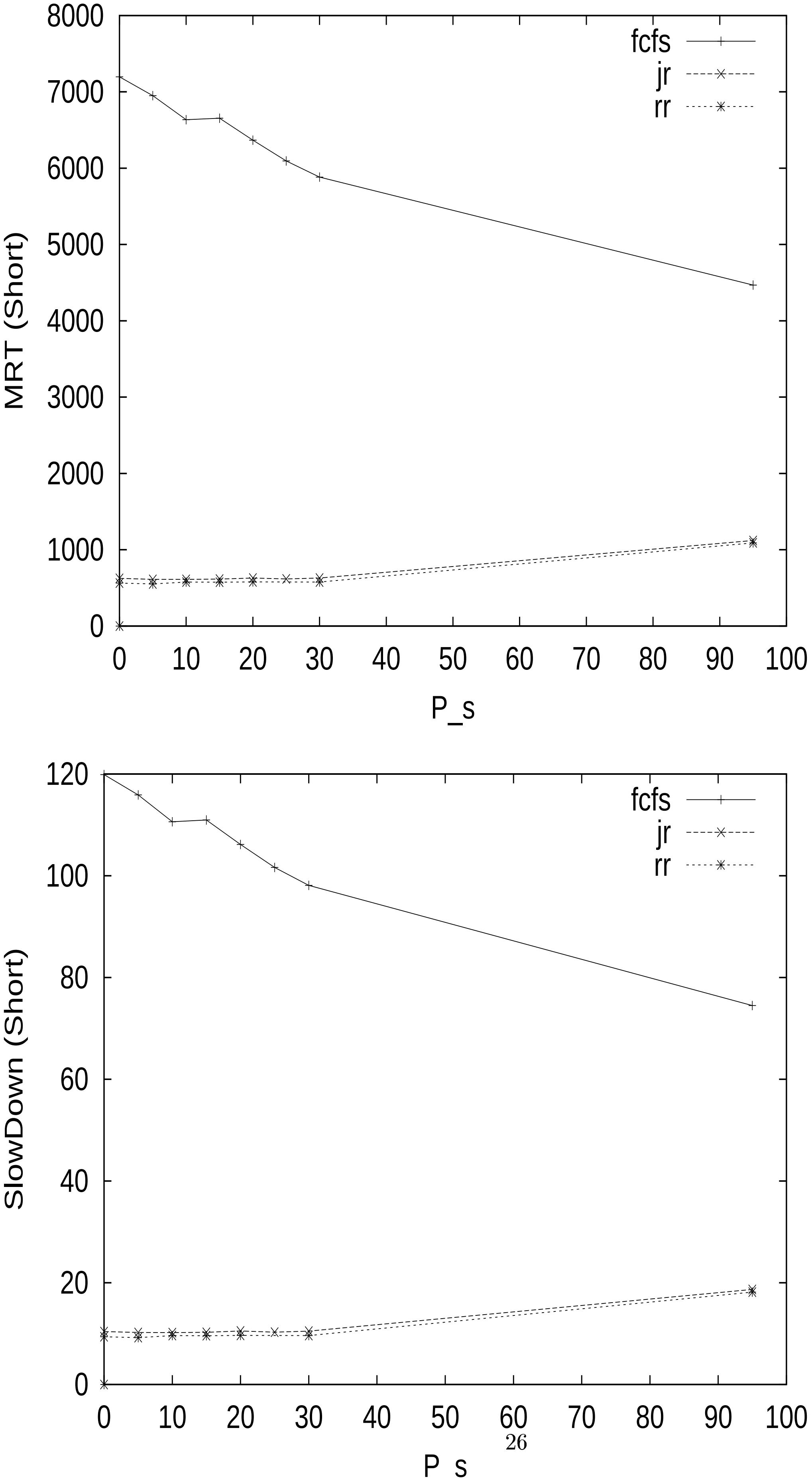

Figure 15: Two Class Normal: Short Jobs Only

[19] M. Livny, J. Basney, R. Raman, and T. Tannenbaum. Mechanisms for high throughput computing. *SPEEDUP Journal*, 11(1), June 1997.

[20] M. Mutka and M. Livny. Profiling workstations' available capacity for remote execution. Technical Report 697, Computer Sciences Department, University of Wisconsin, May 1987.

[21] M. Mutka and M. Livny. The available capacity of a privately owned workstation environment. *Performance Evaluation*, 12:269–284, 1991.

[22] M. W. Mutka and M. Livny. Scheduling remote processing capacity in a workstation-processing bank computing system. In *Proc. of 7th International Conference of Distributed Computing Systems (ICDCS)*, pages 2–9, September 1987.

[23] T. Narten and R. Yavatkar. Remote memory as a resource in distributed systems. In *Proc. of WWOS-III*, pages 132–6, April 1992.

[24] J. Pruyne and M. Livny. Providing resource management services to parallel applications. In *Proc. of the Second Workshop on Environments and Tools for Parallel Scientific Computing*, May 1994.

[25] J. Pruyne and M. Livny. Interfacing condor and pvm to harness the cycles of workstation clusters. *Journal on Future Generations of Computer Systems*, 12, 1996.

[26] J. Pruyne and M. Livny. Managing checkpoints for parallel programs. In *Workshop on Job Scheduling Strategies for Parallel Processing, IPPS 96*, 1996.

[27] R. Raman, M. Livny, and M. Solomon. Matchmaking: Distributed resource management for high throughput computing. In *Proc. of the 7th IEEE International Symposium on High Performance Distributed Computing*, July 1998.

[28] P. Sarkar and J. Hartman. Efficent cooperative caching using hints. In *Proc. of OSDI'96*, 1996.

[29] G. Voelker, E. Anderson, T. Kimbrel, M. Feeley, J. Chase, A. Karlin, and H. Levy. Implementing cooperative prefetching and caching in a globally-managed memory system. In *Proc. of ACM SIGMETRCS 1998*, pages 33–43, 1998.

[30] S. Zhou, J. Wang, X. Zheng, and P. Delisle. Utopia: A load sharing facility for large, heterogeneous distributed computing systems. Technical Report CSRI-257, Computer Systems Research Institute, University of Toronto, 1992.