International Journal of Information, Control and Computer Sciences
ISSN: 2517-9942
Vol:3, No:12, 2009

Achieving fair share objectives via goal-oriented parallel computer job scheduling policies

Sangsuree Vasupongayya

***Abstract*—**Fair share is one of the scheduling objectives supported on many production systems. However, fair share has been shown to cause performance problems for some users, especially the users with difficult jobs. This work is focusing on extending goal-oriented parallel computer job scheduling policies to cover the fair share objective. Goal-oriented parallel computer job scheduling policies have been shown to achieve good scheduling performances when conflicting objectives are required. Goal-oriented policies achieve such good performance by using anytime combinatorial search techniques to find a good compromised schedule within a time limit. The experimental results show that the proposed goal-oriented parallel computer job scheduling policy (namely Tradeoff-fs(Tw:avgX)) achieves good scheduling performances and also provides good fair share performance.

***Keywords*—** goal-oriented parallel job scheduling policies, fair share.

I. Introduction

FAIR share is one of many main objectives supported on several parallel production job schedulers [1,2,3,4,5,6,7,8]. However, these schedulers adopted the idea of fair share from the Fair Share Scheduling [9] proposed for time-sharing systems. A previous study [10] shows that fair share can cause performance problems for some users, especially users with mixtures of jobs due to the priority mechanism currently implemented on many schedulers. This work is proposed to extend goal-oriented parallel computer job scheduling policies to improve such problems. The goal-oriented parallel computer job scheduling policies have been shown to achieve good scheduling performances even when conflicting objectives are required [11,12,13]. Since fair share objective can conflict with other scheduling objectives, applying goal-oriented policies on such problems could be a good solution.

The remaining of this paper is organized as follows. In Section II, goal-oriented parallel computer jobs scheduling policies are reviewed. The current performance problems caused by fair share feature implemented in production schedulers are discussed in Section III. The experimental designs in this work including workloads, policies and performance measures are described in Section IV, while the experimental results and discussions are presented in Section V. Finally, conclusions are given in Section VI.

S. Vasupongayya is with the Department of Computer Engineering, Prince of Songkla University, Hat Yai, Songkhla, 90110, Thailand (e-mail: vsangsur@coe.psu.ac.th phone: 66-74-287360; fax: 66-74-212895).

II. Goal-oriented parallel computer job scheduling

Goal-oriented parallel computer job scheduling policies [11,13] have been recently proposed to reduce the system administrator tasks of adjusting and tuning low-level scheduling parameters for performance by employing a complete search technique to find a 'good' solution in a limited time. A given set of objectives required at a production parallel computer center can be conflicting with each other such as preventing starvation and minimizing average performance. To prevent starvation, for example, difficult jobs (i.e., large jobs, long jobs and large-and-long jobs) must have a high priority because these jobs are likely to be delayed. To minimize average wait time, on the other hand, small and short jobs must have a high priority because majority of jobs are in this category. To achieve such objectives, most scheduling policies use some kind of priority based mechanisms to prioritize and consider jobs for executions according to the priority order. Typically, the priority is implemented by using either a queue-based or a job-based priority mechanism.

Under a queue-based scheme [1,2,3], each job is assigned to a queue according to the job characteristic. For example, a difficult job is assigned to a high priority queue because it is difficult to schedule such job; on the other hand, a short job is assigned to a medium high priority queue because it should not wait too long; a small job is assigned to a low priority queue because it is easy to backfill such job. The queue-based priority job scheduler then selects jobs from each queue according to the priority in the queue and uses small jobs to backfill on available resources to improve the utilization.

Under a job-based scheduling policy [5,6,7], however, a job is prioritized by a weighted function of a set of job measures. For example, to prevent starvation the wait time of each job is added to the priority function. To improve the average measures, the short jobs must be favored. Therefore, the job runtime information is added to the priority function. Each measure has an associated weight value so that the priority function returns a single priority value. The scheduler is then considering jobs for executions according to their priority value.

The priority mechanism either queue-based or job-based implementation can usually achieve only one objective. Goal-oriented parallel computer job scheduling policies have been

shown to achieve good performances on two conflicting objectives by automatically searching for a 'good' compromised schedule. Furthermore, an anytime systematic searching algorithm employed by the goal-oriented parallel computer job scheduling policies always guarantees to find an equal to or a better schedule when more time is given.

In this work, the goal-oriented parallel computer job scheduling policies are extended to cover fair share objective. The scheduling performance and fair share performances are presented in Section V.

III. Fair Share

Fair share is one of the main objectives supported on many production parallel computer schedulers [1,4,6,7,8]. When a fair share policy is in used, each user will have his/her fair share priority dynamically computed. All jobs that belong to a user are given the same fair share priority. A job priority will be adjusted up or down according to the fair share priority of its owner. The fair share priority of each user is a function of the entitled share and the cumulated usage. The entitled shares define the importance of each user relative to other users. The cumulated usage of each user, on the other hand, is the amount of resources that the user has currently used so far. Both the entitled share and the cumulated usage are dynamically calculated within a fair share window. This type of fair share is referred to as a relative fair share model. One important configurable parameter of fair share is a fair share window which is the period of time where the usage of each user is cumulated. The typical default fair share window size is one day or seven days.

A previous study [14] shows that fair share feature does not affect the average performance; however, this study does not consider per-user performance or maximum wait performance. Another study [10] shows that the fair share is in fact fair because it can prevent heavy-demand users to dominate the system resources. As a result, resources are available for other users. In addition, the later study considers the per-user performance and the finding demonstrates that fair share objectives can affect performance of some users severely due to the non-preemptive nature of the underline system and the use of priority mechanisms.

These users suffer poor performance because their difficult jobs (i.e., large jobs, long jobs or both large and long jobs) are delayed in reserving recourses because their difficult jobs enter the system behind their not-so-difficult jobs. Due to the priority mechanism, once a job of a user is started, the fair share priority of the user is lowered which may prevent the difficult jobs of the same user from receiving a reservation. These difficult jobs are in nature required a reservation to reserve enough resource for their executions. The delay in reserving resources for such jobs can cause an extended delay of starting these jobs.

In this work, goal-oriented policies are applied to improve the problems. To do so, a newly proposed goal-oriented parallel computer job scheduling policy is proposed. Next, the experimental settings including the policies, the workloads, the performance and fair share measures are given.

IV. Experimental Design

All policies are evaluated using an event-driven simulator with a real job trace from a production parallel computer center. The job trace is a ten-monthly workload that ran on an Intel Itanium Linux cluster (IA-64) at the National Center for Supercomputing Applications at the University of Illinois at Urbana-Champaign during June 2003 to March 2004.

TABLE I
INFORMATION OF NSC IA-64 WORKLOAD

	Proc.			#jobs per users		
Month	demand	#users	#jobs	Avg.	Median	Max.
6/03	82%	73	2191	30.0	8.0	659
7/03	89%	68	1400	20.6	8.0	145
8/03	79%	73	3221	44.1	8.0	1873
9/03	72%	74	3057	41.3	15.0	703
10/03	71%	75	4149	55.3	15.0	1151
11/03	73%	81	3443	42.5	17.0	665
12/03	74%	61	3521	57.7	14.0	635
1/04	73%	53	3156	59.5	17.0	679
2/04	74%	73	3969	54.4	28.0	541
3/04	75%	70	3466	49.5	15.5	1234
	Job size (NT)			demand (NT) per user		
Month	Avg.	Median	Max.	Avg.	Median	Max.
6/03	34.5	0.8	960.0	1034.7	24.0	24071
7/03	60.6	1.3	1536.0	1247.4	145.7	16719
8/03	23.4	0.0	1536.0	1031.6	120.0	14346
9/03	21.7	0.1	912.0	895.5	72.5	18499
10/03	16.3	0.4	912.0	899.5	114.7	8060
11/03	19.5	0.7	1536.0	827.1	27.6	10183
12/03	20.1	1.1	1152.0	1159.3	23.2	17776
1/04	22.1	5.1	1920.0	1313.6	317.4	10340
2/04	16.6	0.3	1824.0	900.3	93.3	8931
3/04	20.6	0.0	1832.8	1018.0	46.1	12892
	Job size (N)			Job size (T)		
Month	Avg.	Median	Max.	Avg.	Median	Max.
6/03	12.1	4.0	128	1.4h	0.20h	12h
7/03	23.5	8.0	128	1.9h	0.18h	12h
8/03	7.3	1.0	128	1.1h	0.00h	12h
9/03	9.1	1.0	128	1.4h	0.03h	12h
10/03	5.0	1.0	128	2.0h	0.13h	12h
11/03	6.0	1.0	128	2.5h	0.15h	12h
12/03	5.6	1.0	128	3.6h	0.33h	24h
1/04	10.7	2.0	128	4.5h	1.10h	24h
2/04	5.0	2.0	128	3.1h	0.11h	24h
3/04	5.8	1.0	128	2.4h	0.00h	24h

Scheduling performances of each month are computed for jobs submitted during the month. To be realistic, however, each simulation of a given month includes a one-week (from the previous month) warm up and a cool-down period in which jobs (from the next month) continue to arrive. The cool-down period is typically a few days only because the period will end when all jobs submitted during the month for which the performance measures are computed have all started. To understand the full potential of all policies, the simulator uses the perfect runtime information of each job, and thus the impact of inaccurate runtime estimates is eliminated.

Table 1 shows the workload characteristic in each month including demand (Proc. demand), number of users (#users), number of jobs (#jobs) and job size (i.e., NT: node-hour, N: node, T: runtime in hours). Job size information is given in

three measures including average (Avg.), median and maximum (Max.) values.

A. Policies

Two types of policies are evaluated in this work. The first type of policies is the simple priority-based scheduling policies with backfill techniques. This set of policy includes the basic First-Come-First-Serve (FCFS) priority, Largest-Slowdown-First (LXF) priority and the fair share priority. The second type of policies is the goal-oriented parallel computer job scheduling policies both the original version and the newly proposed version.

The most widely used non-preemptive parallel job scheduling policies are priority based scheduling policies with backfill techniques. Backfill techniques allow lower priority jobs to start on idle resources as long as their executions do not delay the scheduled start time of the oldest waiting job. In this work, FCFS-backfill and LXF-backfill policies are used as baseline policies because FCFS-backfill tends to achieve good maximum wait, whereas LXF-backfill achieves good average performance.

A simple relative fair share policy namely RelShare(1d) is also evaluated. RelShare(1d) policy uses one-day fair share window. Under this policy, the job is ordered by its owner fair share priority. The fair share priority of each user is dynamically computed based on the user's cumulated usage during the current fair share window (i.e., the current day). The cumulated usage of a user includes the actual usage of the user from the beginning of the day until the current time as well as the expected usage of the currently running jobs and the job with the reservation (if any). However, the usage is only computed within the current fair share window. That is, if the job is expected to run across fair share window, only the usages in the current window are cumulated.

Goal-oriented parallel computer job scheduling policies, namely Tradeoff(Tw:avgX) [11,13], are extended to cover fair share objective by using the fair share priority to organize the search space. That is, the fair share priority of the job owner is used as a branching heuristic. This way, the fair share objective biases the searching process to the schedule that favors fair share objectives. The way in which the fair share priority is calculated here is similar to that in the RelShare(1d) policy. That is, the fair share window is one-day and the cumulated usages include the actual usages and the expected usages in the current fair share window. This adapted policy is called Tradeoff-fs(Tw:avgX). Under this policy, the searching process starts from a fair share favored schedule and continue its search to other schedules. During the searching process the current 'best' schedule found so far is kept. To find the 'best' schedule, the searching algorithm compares the newly found schedule with the current 'best' schedule. If the new schedule results in a 'better' performance according to the current objectives, the new schedule is kept as the current 'best' schedule. The searching process continues until the time limit is reached. Note that the definition of the 'best' performance can be found in the previous works [11,15] and the left-most path schedule is always started as the first current 'best' schedule.

B. Performance Measures

Both widely used scheduling performances and a fair share performance are studied in this work. Previous work [9] has shown that achieving fair share can affect the scheduling performance of some users. Thus, several widely used scheduling performance measures [16,17,18,19,20] are studied. These measures include average wait, average bounded slowdown, maximum wait and 99th-percentile wait. The maximum wait and the fair share performance measures are focused in this study because previous studies indicated that the fair share has little or no impact on average performance [14] but the maximum wait performance is affected severely [10].

The slowdown is the ratio of the job turnaround time to its runtime. However, if the job is very short, its slowdown measure is very high. Thus, the performance of a few very short jobs can affect the overall slowdown measure. To minimize this impact, a bounded slowdown is used instead of the slowdown. The bound in this study is set to one minute meaning that any job shorter than one minute will calculate its slowdown as a one-minute job.

To evaluate the fair share performance, the deviation in node hours (dev)—defined in [10] is used for evaluating the per-user fair share performance of each policy. The dev measure is designed to be the difference between the cumulated actual usage and the cumulated entitled share of each user over a given fair share window. In this work, a fair share window of one-day which is a normal default fair share window size of many production job schedulers is used. By definition, a positive dev value and a negative dev value mean an over-share and an under-share, respectively.

V. Results and Discussion

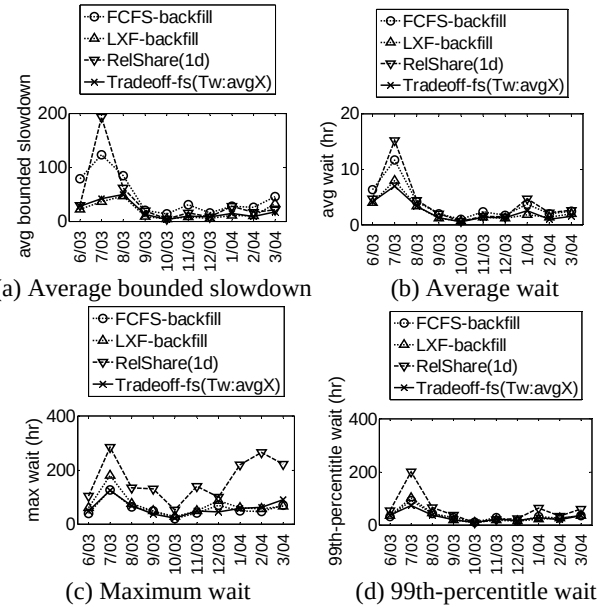
Results and discussions are organized as follows. First, the overall scheduling performance of the all policies studied is presented. Next, the fair share performance is given. Then, the detailed analysis of how the proposed goal-oriented policy can achieve good scheduling performances, while maintaining good fair share performance is discussed.

A. Overall Scheduling Performance

The overall scheduling performances of FCFS-backfill, LXF-backfill, RelShare(1d), and Tradeoff-fs(Tw:avgX) are presented in Figure 1. Figure 1(a)-(d) show the average bounded slowdown, the average wait, the maximum wait, and the 99th-percentiel wait performance, respectively. As expected, FCFS-backfill provides good maximum wait performances on all months (Figure 1(c)), while LXF-backfill provides good performances on both average bounded slowdown Figure 1(a)) and average wait (Figure 1(b).

Impressively, Tradeoff-fs(Tw:avgX) achieves the best or close to the best performance on all measures in all months, except perhaps a slightly high maximum wait in March 2004.

These results indicate that the proposed goal-oriented parallel job scheduling policy does not degrade the scheduling performances. Note that Tradeoff-fs(Tw:avgX) significantly outperforms RelShare(1d) on maximum wait measure.



(a) Average bounded slowdown (b) Average wait

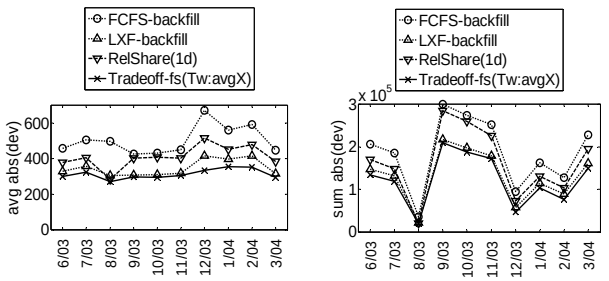
(c) Maximum wait (d) 99th-percentitle wait

Fig. 1 Overall scheduling performance

In conclusion, the results in this section show that the proposed goal-oriented parallel computer job scheduling policy performs well on all scheduling performance measures. The next section investigates the fair share performance of each policy.

B. Fair share performance

This section is focusing on evaluating the fair share performance of each policy by showing the deviation of node hour performance. Figure 2 shows the average over all users and the sum of all users of the absolute per-user dev in each of the ten month of each policy.



(a) Average absolute dev (b) sum absolute dev

Fig. 2 Monthly dev performances

Figure 2(a) shows the average absolute dev over all users in each month of the four policies studied, whereas Figure 2(b) shows the total dev of all users in each month. The average dev performance in Figure 2(a) shows that the proposed Tradeoff-fs(Tw:avgX) policy outperforms all policies and achieves the lowest deviation of fair share among users in all months. RelShare(1d) policy reduces unfairness observed under FCFS-backfill policy but it still loose to LXF-backfill policy. Note that LXF-backfill provides good fair share performances but it achieves these performances by favoring the majority of jobs, resulting in poor overall maximum wait performances. Tradeoff-fs(Tw:avgX) policy, on the other hand, achieves good fair share performances without degrading the maximum wait performance.

According to the results so far, the conclusion is that Tradeoff-fs(Tw:avgX) not only achieves good scheduling performances but also provides good fair share performances.

C. How goal-oriented policies solve unfairness problems

In this section, the per-user performance is analyzed to discover how Tradeoff-fs(Tw:avgX) achieves such good performances. To answer the question, the performance of the proposed Tradeoff-fs(Tw:avgX) policy is compared against the performance of the original Tradeoff(Tw:avgX). This result provides evidence that the fair share priority used as a branching heuristic in the newly proposed policy does influence the good performances. In addition, the original Tradeoff(Tw:avgX) has never been evaluated on any fair share measure. Figure 3 shows performances of the two goal-oriented parallel computer job scheduling policies and the RelShare(1d) policy.

Figure 3(a)-(f) show the average bounded slowdown, the average wait, the maximum wait, the 99th-percentile wait, the average absolute per-user dev and the sum absolute per-user dev performances of RelShare(1d), Tradeoff(Tw:avgX) and Tradeoff-fs(Tw:avgX), respectively. The results show that Tradeoff-fs(Tw:avgX) outperforms Tradeoff(Tw:avgX) on fair share measures (Figure 3(e)-(f)), while achieves similar scheduling performances (Figure 3(a)-(d)).

Next, the users who suffer under RelShare(1d) as reported in [9] are analyzed. Table II shows the dev, maximum wait, average wait and average bounded slowdown performances under each policy of the users with worse degradation in wait time under RelShare(1d) in July 2003 month. Results on other months are similar. The results on July 2003 are presented here due to its high load characteristic.

As shown in Table III, Tradeoff-fs(Tw:avgX) policy improves performances of the users (except user #21) who suffer under RelShare(1d) as being seem from the dev measure. User #21 is slightly under-shared under Tradeoff-fs(Tw:avgX) policy. However, all scheduling performances of user #21 under Tradeoff-fs(Tw:avgX) policy are better than those under RelShare(1d) policy.

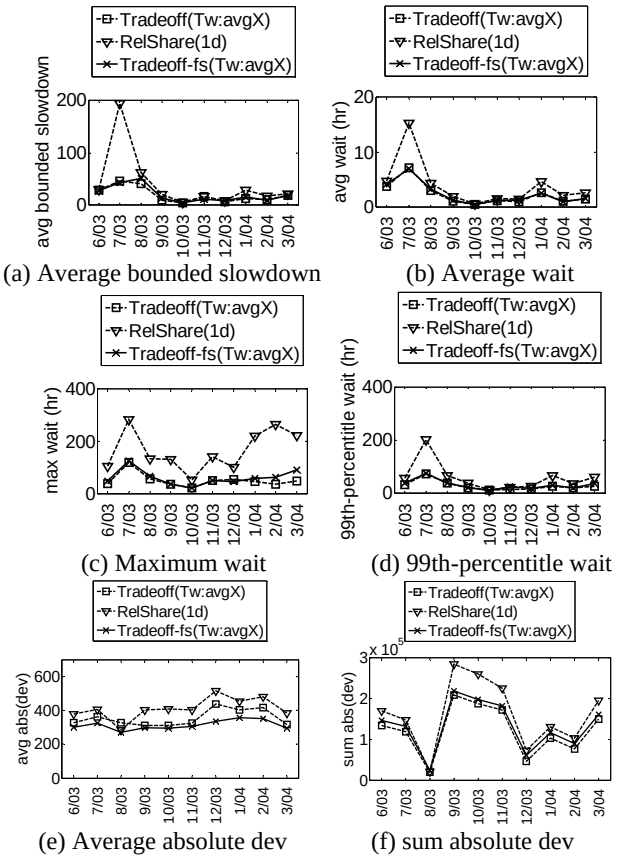
To understand how Tradeoff-fs(Tw:avgX) policy achieves good performances, Figure 4 shows the number of jobs belonging to user #8 on each day in July 2003 month. The numbers of three types of jobs are shown in the figure: the jobs that are waiting at the beginning of that day, the jobs that are submitted during that day, and the jobs that are started on that day. Figure 4(a) shows the job information of user #8

under RelShare(1d) policy, while Figure 4(b) shows the same information under Tradeoff-fs(Tw:avgX) policy. User #8 suffers poor performance under RelShare(1d) because only a few jobs of the user can start each day. On the other hand, Tradeoff-fs(Tw:avgX) policy allows several jobs of user #8 to start each day. Thus, the backlog under Tradeoff-fs(Tw:avgX) policy is smaller than that under RelShare(1d). To further illustrate this point, Figure 5(a) shows the per-day demand, usages and entitle shares of user #8 under RelShare(1d) policy, while Figure 5(b) shows the same information under Tradeoff-fs(Tw:avgX) policy. The average demand of all active users is also given for comparison purposes. Since the backlog is reduced under Tradeoff-fs(Tw:avgX) policy the demand on each day under Tradeoff-fs(Tw:avgX) is lower than that under RelShare(1d) policy. Note that the demand is computed daily. Thus, the demand on each day consists of the newly submitted jobs and the current backlog on that day. This way, some jobs can be counted as demand on several days if the jobs remain in the system for several days.

The results so far have shown that the branching heuristic used by Tradeoff-fs(Tw:avgX) policy helps to improve the fair share performance while providing the similar scheduling performances. Tradeoff-fs(Tw:avgX) policy also improves the performance of user with mixtures of jobs who suffers poor maximum wait performance under RelShare(1d) policy. The explanation here is that Tradeoff-fs(Tw:avgX) policy allows jobs of these users to start as long as their executions result in a better scheduling performance. On the other hand, RelShare(1d) policy blocks the jobs of these users because of the priority mechanism.

Performances of other users under Tradeoff-fs(Tw:avgX) policy are also improved as shown in Table III. Table III shows the dev, maximum wait, average wait and average bounded slowdown of users with mixture of jobs but spread their jobs out over a period of time during the month. These users are not suffered under RelShare(1d) policy. In fact, they could benefit under RelShare(1d) policy by becoming over-share users. For example, user #70 on July 2003 goes from under-share of 1040.8 node-hours under FCFS-backfill policy to over-share of 423.5 node-hours under RelShare(1d) policy and user #103 on November 2003 goes from under-share of 355.1 node-hours under FCFS-backfill policy to over-share of 424.8 node-hours under RelShare(1d) policy.

Tradeoff-fs(Tw:avgX) policy, on the other hand, does not allow these users to get benefits from the fair share. User #70 on July 2003 is under-share by only 89.2 node-hours and user #103 on November 2003 is under-share by only 38.1 node-hours, while these two users become over-share users under RelShare(1d) policy. In addition, Tradeoff-fs(Tw:avgX) policy also reduce the amount of under-share of the already under-share users such as user #113 on August 2003. The amount of under-share of this user is reduced from 1178.6 node-hours under FCFS-backfill policy to under-share of only 303.2 node-hours under Tradeoff-fs(Tw:avgX), whereas this user is still under-share of 411.5 node-hours under RelShare(1d).



(a) Average bounded slowdown (b) Average wait

(c) Maximum wait (d) 99th-percentitle wait

(e) Average absolute dev (f) sum absolute dev

Fig. 3 Scheduling and fair share performance of RelShare(1d), Tradeoff(Tw:avgX) and Tradeoff-fs(Tw:avgX)

VI. Conclusions

This study presents and analyzes the results of extending a goal-oriented parallel job scheduling policy to cover fair share objective by applying the fair share priority as a branching heuristic. All policies are evaluated using an event-driven simulator with a ten-monthly workload that ran on a real production computer center. In addition to the fair share objective, the goal-oriented policies considered two objectives—preventing starvation and minimizing average measures. The scheduling performance of the proposed policy (i.e, Tradeoff-fs(Tw:avgX) is compared against the two priority backfill policies namely FCFS-backfill and LXF-backfill. These policies have been shown to achieve the best performance on each objective studied. In addition, a simple relative fair share policy namely RelShare(1d) policy is also evaluated.

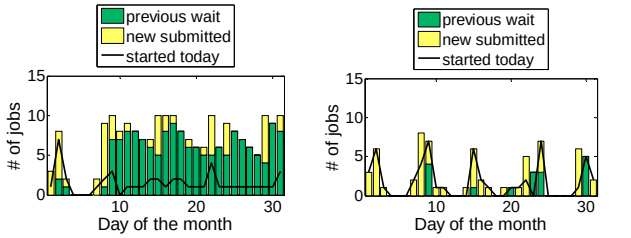
The experimental results show that the proposed goal-oriented parallel computer job scheduling policy (i.e., Tradeoff-fs(Tw:avgX)) does achieve the best or close to the best performance on all scheduling objectives, while provides a good fair share performance. Tradeoff-fs(Tw:avgX) policy works similar to RelShare(1d) policy such that it does not allow heavy-demand users to overtake the system resources. However, Tradeoff-fs(Tw:avgX) policy does not cause any

scheduling and fair share performance problems for users with mixture of jobs as observed under RelShare(1d) policy. In fact, Tradeoff-fs(Tw:avgX) policy is more fair because it optimizes the dev measure better than the RelShare(1d) policy.

The results presented in this work are very encouraging even though the Tradeoff-fs(Tw:avgX) policy does not consider the fair share directly as parts of its objectives. In the future work, low overhead fair share measures will be investigated so that the fair share can be directly considered as one of the objectives in the goal-oriented parallel computer job scheduling policies.

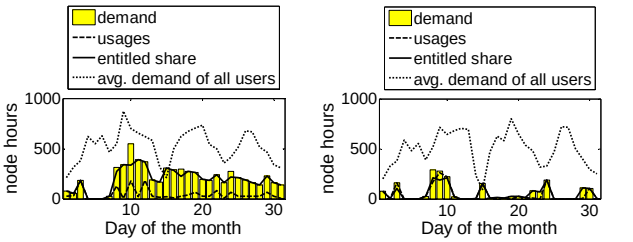
TABLE II
PERFORMANCE OF USERS WITH WORSE DEGRADATION IN WAIT TIME UNDER RELSHARED(1D) IN JULY 2003

Deviation in node hours				
User	FCFS-bf	LXF-bf	RelShare	Tradeoff-fs
8	-2907.8	-1606.5	-7245.0	-1396.0
21	-1706.0	-1611.1	-667.8	-2311.1
24	-1787.6	-894.3	-4592.9	-767.9
49	1532.0	-1373.0	68.6	190.0
71	6585.3	3803.6	4582.9	3712.5
Maximum wait				
User	FCFS-bf	LXF-bf	RelShare	Tradeoff-fs
8	82.8h	26.6h	280.7h	61.0h
21	65.6h	63.5h	109.5h	68.4h
24	114.8h	30.7h	283.8h	33.2h
49	66.2h	179.8h	163.5h	84.0h
71	105.1h	143.5h	211.2h	125.2h
Average wait				
User	FCFS-bf	LXF-bf	RelShare	Tradeoff-fs
8	21.1h	4.6h	80.7h	10.1h
21	11.4h	6.3h	11.9h	6.7h
24	26.1h	8.8h	73.3h	10.7h
49	11.5h	16.0h	23.7h	8.0h
71	29.1h	27.7h	31.8h	22.0h
Average bounded slowdown				
User	FCFS-bf	LXF-bf	RelShare	Tradeoff-fs
8	168.1	34.7	907.2	89.6
21	126.6	28.9	103.8	35.6
24	459.5	122.4	2033.2	181.2
49	106.7	42.6	108.4	14.7
71	148.2	45.8	94.1	103.6



(a) RelShare(1d) (b) Tradeoff-fs(Tw:avgX)

Fig. 4 Per-day number of jobs of user #8 on July 2003



(a) RelShare(1d) (b) Tradeoff-fs(Tw:avgX)

Fig. 5 Per-day node-hours of user #8 on July 2003

TABLE III
PERFORMANCE OF USERS WITH MIXTURE OF JOBS BUT SPREAD THEIR JOBS

Dev					
User	Mo	FCFS-bf	LXF-bf	RelShare	Tradeoff-fs
70	7/03	-1040.8	-581.3	423.5	-89.2
113	8/03	-1178.6	-351.6	-411.5	-302.2
103	11/03	-355.1	-483.0	424.8	-38.1
Maximum wait					
User	Mo	FCFS-bf	LXF-bf	RelShare	Tradeoff-fs
70	7/03	49.0h	60.0h	20.1h	34.0h
113	8/03	35.6h	28.7h	30.6h	29.8h
103	11/03	33.9h	32.4h	20.5h	38.8h
Average wait					
User	Mo	FCFS-bf	LXF-bf	RelShare	Tradeoff-fs
70	7/03	25.7h	17.4h	8.9h	12.0h
113	8/03	19.2h	11.4h	12.4h	10.8h
103	11/03	8.9h	2.9h	1.1h	1.6h
Average bounded slowdown					
User	Mo	FCFS-bf	LXF-bf	RelShare	Tradeoff-fs
70	7/03	54.1	25.8	28.8	31.4
113	8/03	59.9	15.9	61.4	18.1
103	11/03	141.7	27.8	2.8	7.0

REFERENCES

[1] OpenPBS, http://www.nas.nasa.gov/Software/PBS/
[2] PBS pro, http://www.pbspro.com
[3] LSF, http://www.platform.com/product/ lsffamily.
[4] LSF fair share documentation, http://accl.grc.nasa.gov/job_schedulers/lsf/ Docs/lsf6.1/lsf6.1_admin /E_fairshare.html
[5] D. Jackson, Q. Snell & M. Clement. "Core algorithms of the MAUI scheduler". In proceeding of the Workshop on Job Scheduling Strategies for Parallel Processing, 2001.
[6] Maui scheduler, http://www.supercluster.org/maui
[7] Moab scheduler, http://www.clusterresources.com/products/mwm/docs/moabadminguide450.pdf
[8] S. Kannan, M. Roberts, P. Mayes, D. Brelsford & J. Skovira. "Workload management with LoadLeveler". Technical Report, IBM Redbook, 2001.
[9] J. Key & P. Lauder. "A fair share scheduler". Communications of the ACM, 31(3):44-55, 1988.
[10] S. Vasupongayya, "Impact of fair share and its configurations on parallel job scheduling algorithms". (to appear). In proceeding of the 2009 WASET International Conference on High Performance Computing, Venice, Italy, October 2009.
[11] S.-H. Chiang and S. Vasupongayya, "Design and potential performance of goal-oriented job scheduling policies for parallel computer workloads". In the IEEE Transaction on Parallel and Distributed Systems. 19(12):1642-1656, 2009.
[12] S. Vasupongayya, "Goal-oriented parallel job scheduling: A revisit", In proceeding of the 2nd UBU-Research, Ubonratchathani, Thailand, July 2008.

[13] S. Vasupongayya, S.-H Chiang and B. Massey, "Search-based job scheduling for parallel computer workloads", In proceeding of the IEEE Cluster, Boston, MA, 2005.
[14] S. Kleban and S. Clearwater. "Fair share on high performance computing system: What does fair really mean?" in proceeding of the IEEE International Symposium on Cluster Computing and the Grid, 2003.
[15] S. Vasupongayya and S.-H. Chiang. "Multi-objective models for scheduling jobs on parallel computer systems". In proceeding of IEEE Cluster, Barcelona, Spain, 2006.
[16] S.-H. Chiang, A. Arpaci-Dusseau and M. Vernon. "The impact of more accurate request runtimes on production job scheduling performance". In Lecture Notes in Computer Science (2537):103-127, 2002.
[17] S.-H. Chiang and C. Fu. "Benefit of limited time-sharing in the presence of very large parallel jobs". In proceedings of the IEEE International Parallel and Distributed Processing Symposium, 2005.
[18] S.-H. Chiang and M. Vernon. "Production job scheduling for parallel shared memory systems". In proceeding of the IEEE International Parallel and Distributed Processing Symposium, 2001.
[19] D. Talby and D. Feitelson, "Supporting priorities and improving utilization of the IBM SP2 scheduler using slack-based backfilling". In proceeding of the International Parallel Processing Symposium, 1999.
[20] D. Talby and D. Feitelson, "Improving and stabilizing parallel computer performance using adaptive backfilling". In proceeding of the IEEE International Parallel and Distributed Processing Symposium, 2005.

Sangsuree Vasupongayya received a Bachelor of Engineering in Computer Engineering from Prince of Songkla University, a Master of Science in Computer Science from California State University Chico and a Ph.D. degree in Computer Science from Portland State University. Currently, Dr. Vasupongayya is the associate department head for academic affairs at the computer engineering department, Faculty of Engineering, Prince of Songkla University. Interested research areas include high-performance computer resource scheduling, cryptography, eLearning and engineering curriculum and education.