Adaptive Metric-Aware Job Scheduling for Production Supercomputers

Wei Tang, Dongxu Ren
Illinois Institute of Technology
Chicago IL, USA
{wtang6, dren1}@iit.edu

Narayan Desai
Argonne National Laboratory
Argonne IL, USA
desai@mcs.anl.gov

Zhiling Lan
Illinois Institute of Technology
Chicago IL, USA
lan@iit.edu

ABSTRACT

Job scheduling is a critical and complicated task on large-scale supercomputers where a scheduling policy is expected to fulfill amorphous and sometimes conflicting goals from both users and system owners. Moreover, the effectiveness of a scheduling policy is dependent on workload characteristic which varies from time to time. Thus it is not easy to design a versatile scheduling policy being effective under all circumstances. To address this issue, we propose an adaptive metric-aware job scheduling strategy. First, we propose metric-aware scheduling which enables the scheduler to balance the scheduling goals based on different interested metrics such as job waiting time, fairness, and system utilization. Second, we enhance the existing scheduler to be able to adaptively adjust scheduling policies based on the feedback of the metrics under real-time monitoring. We evaluate our strategy using real workload from supercomputer centers and the experimental results show our methods can effectively improve system performance.

Keywords

job scheduling, resource management, supercomputing, metric

1. INTRODUCTION

Job scheduling is a critical task for large-scale supercomputers. The effectiveness of a job scheduling policy directly influences the satisfaction of both users or system owners. For users, fast job turnaround and fairness are the major interested metrics, while for system owners, the system utilization is as important. Moreover, large production computing centers now face an increasing set of considerations in scheduling, such as avoiding failure interrupts and energy efficiency. All these considerations (or metrics) are correlated but sometimes conflicted. Even worse, the priorities differ from site to site and from time to time, which complicated the design of a versatile job scheduling policy.

HPDC '12 Delft, the Netherlands

Traditional scheduling policies can serve certain purposes but not balance them well. For example, using "first-come, first served (FCFS)" can achieve strict user fairness but not good for response time and also cause system fragmentation. On the other hand, using "short-job first (SJF)" can achieve best response time but violates job fairness and will likely cause job starvation. Some policy like "max expansion factor first" can act in the middle but also cannot balance the wanted metrics at will.

Meanwhile, the user benefit and system cost are not considered as a whole. Typically, in decision making, job prioritizing and resource allocation are separated into two subsequent phrases. This causes the latter has much fewer options to achieve more reasonable decisions. For example, when a first-priority job has no enough nodes to run, it reserve some resource while draining some idle resources, which may be used by the second job being scheduled instead. This kind of resource draining causes external fragmentations. Backfilling can reduce fragmentation but it is only a mitigation action after those fragmentation already generated by job prioritizing [22].

Further, scheduling policy heavily depends on the workload characteristic. Event hough we can identify a policy to achieve our integrated interests well, a different kind of workload may invalid it entirely. Although event-driven simulators can help to test a policy over a historical workload [23], it cannot enable the scheduler adapt scheduling policies to the changing situation efficiently.

Motivated by these issues, we propose an adaptive metric-aware job scheduling mechanism. Our solutions are two folded. First, we propose a metric-aware job scheduling mechanism which schedules and allocates jobs in an integrated fashion. Specifically, each job's schedule (when and where to run) is correlated with the impact it could make to the metrics we are interested. And the final decision is made to balance different metrics, representing both user benefits and system costs.

Second, we propose a mechanism to adaptively tune scheduling policies based on workload characteristic change. By monitoring the metrics values in real time, the scheduler can adjust its scheduling policy to favor the metrics that are less satisfied recently, thereby mitigating the impact of workload characteristic on the scheduling policy. For example, if in the latest period of time the system utilization rate is reported below a certain threshold (or a longer-term average), the metric-aware scheduling policy will be tuned to favor the metric system utilization more than others. The second mechanism is built on the basis of the first one.

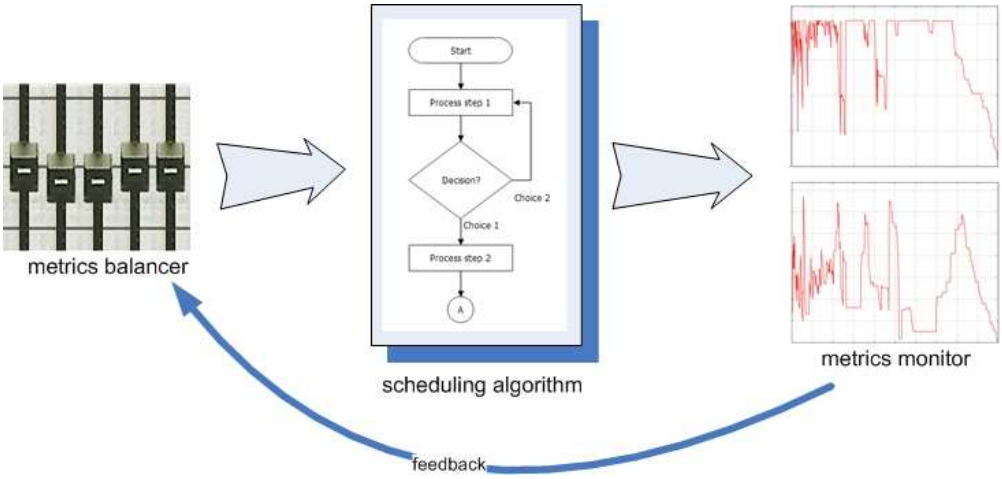


Figure 1: Diagram of adaptive metric-aware job scheduling framework.

We implemented our schedule mechanism in a production resource management system named Cobalt [1]. We evaluated our mechanism using recent real job traces from production Blue Gene/P system at Argonne National Laboratory as well as archived job traces published on Parallel Workload Archive [2]. The experimental results show that our scheduling mechanism is superior in balance user benefit and system costs.Particularly the targeted metrics can be achieved by tuning scheduling policy in advance. Also the adaptive policy tuning is demonstrated effective in adapting to varying workload characteristic.

The remainder of this paper is organized as follows. Section 2 further discusses our motivation and methodology. Section 3 describes the detailed design of the metric-aware job scheduling mechanism. Section 4 describes the detailed design of adaptive policy tuning. Section 5 illustrates our experimental results Section 6 discusses some related work. Section 7 summarizes the paper with some discussion on future work.

2. MOTIVATION AND METHODOLOGY

In job scheduling context, the effectiveness of a job scheduling policy are measured by certain metrics. A metric represents certain resource management goals or interests. Numerals scheduling research has been discussed various scheduling metrics. Generally, based on the category of the interests origin, the metrics can be grouped into two categories: user-centric metrics, reflecting levels of user satisfaction, and system-centric metrics, measuring the system utilization and costs.

Regarding user satisfaction, one of the most commonly used metrics for scheduling is the average response time (the time from when a job was submitted until it terminated) and waiting time(the time from a job's submission to its start). An alternative is the slowdown (the response time normalized by the job's actual running time). Meanwhile, user utility function is also used to associate the user satisfaction with a function over the response time [8][13]. While above metrics are considered reflecting the "efficiency" (meaning few delays and fast turnround), "fairness" is also a major metric. Likely being the reason why queues were formed in the first place, "fairness" may be cared about more by users than actual delays [15].

From system perspective, the system utilization and cost are equally important with user satisfaction. The supercomputers are expensive resources invested to serve applications and it is not desirable to let them idle often. Thus, the system utilization rate (the ratio of busy node-hours to the total node-hours over the examined time period) is a key metric that system owners care about. Because of the scheduling algorithm and workload characteristic, fragmentation is inevitable. How to reduce fragmentation is big challenge to improve system utilization. Loss of capacity (the ratio the idle node-hours while jobs are queuing compared to total capacity) [27][24] is a related metric that can be aimed to improve. Meanwhile, increasing concerns such as power consumption and fault interruptions have related metrics to measure. For example, delivered node-hours per watt can measure the energy efficiency.

All these metrics seems individual but are correlated, or sometimes conflicted. However, existing scheme mechanism has some intrinsic limitations to balance these considerations as a whole. First, the job prioritizing and resource allocation are separated into phrases. That is, the latter have limited choices when allocate jobs.

Even within job prioritizing, different goals are not balanced well. For example, using "first-come, first served (FCFS)" can achieve good user fairness but not good for average response time. On the other hand, using "short-job first" can achieve best response time but violates job fairness.

A mechanism is needed to explicitly balance these considerations. Moreover, this balance is not universal; different systems (or same system during different time) have varied priorities that result in some considerations prevailing over others. Therefore we propose an adaptive metric-aware job scheduling mechanism. The goal of this mechanism is two-fold. First, it shall provide a mechanism to connect job priorities explicitly to the impact (in metric space). Second, it shall be able to tune scheduling policies based on the feedback of monitored metrics.

The diagram of our design is shown in Figure 1, where there are three major components: a metrics balancer, a

scheduling algorithm, and a metrics monitor. The metrics balancer is used to balance different priorities or scheduling goals in composing an integrated scheduling algorithm. By running the scheduling algorithm, relevant metric values will be monitored. The feedback from metrics monitor will be sent back to metrics balancer so that it can adjust the weight of different factors to maintain the desired balance. This feedback process can be conducted manually or automatically. We call the process how the metrics balancer impacts the scheduling algorithm as "metric-aware job scheduling", and the process how the metrics monitor influences the adjustment to metrics balancer as "dynamic policy tuning". In following two sections we will present the detailed design of the metric-aware scheduling and dynamic policy tuning, respectively.

3. METRIC-AWARE JOB SCHEDULING

In this section we will present the detailed design of metric-aware job scheduling. The goal of metric-aware job scheduling is to provide a metric-balancer which can determine a scheduling algorithm that balances the interested metrics. In our current design, the metrics to be balanced are of three aspect. One reflect system utilization and other two reflects user satisfaction: one is "fairness", the other is "efficiency" (i.e. fast turnaround).

As mentioned earlier, the user-centric metrics are mainly influenced by job queuing policies and the system metrics are correlated with job allocation. Existing scheduling scheme allocate jobs one by one in their priority order. The problem is that when allocate one job at one time, it can achieve the best allocation for that single job but neglects the potential influence of other jobs in the queue (one simple example is shown in Figure 2). Thus we propose to schedule and allocate a group of jobs at one time so that the job allocation can result in more optimal system utilization. We call the number of jobs that are scheduled at one time as "window size", which varies from one to up to the number of queued jobs. Intuitively, the larger the window size is, more reasonable job allocation can be achieved but may violate user-centric metrics especially for the fairness.

To balance fairness and efficiency, we sort the queue using priority score factoring both waiting time and running time. By tune the weight of these two parameter, we can balance the priority on fairness and efficiency. The detailed scheduling steps are described as follows.

Step 1: Calculate job i's score regarding waiting time, mapping the values to [0, 100]:

$$S_w = 100 \times \frac{wait_{max}}{wait_i}, \tag{1}$$

where $wait_{max}$ is the maximum waiting time of the current queue and $wait_i$ is the current waiting time of job i.

Step 2: Calculate job i's score regarding requested walltime, mapping the values to [0, 100],

$$S_r = 100 \times \frac{walltime_{max} - walltime_i}{walltime_{max} - walltime_{min}}, \tag{2}$$

where $walltime_{max}$, $walltime_{min}$ are the maximum and minimum walltime times of the current queue, respectively, and $walltime_i$ is the walltime of job i.

Step 3: Calculate job i's balanced priority:

$$S_p = \alpha \times S_w + (1 - \alpha) \times S_r, \tag{3}$$

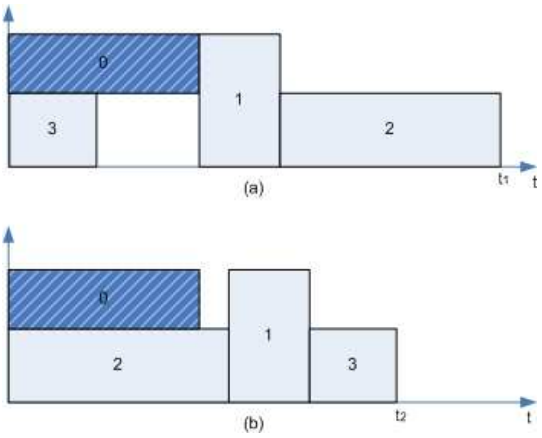


Figure 2: An example showing the limitation of scheduling and allocate jobs one by one. Job 0 is running, Jobs 1, 2, and 3 are waiting . (a) schedule and allocate job one by one in priority order; (b) schedule and allocate in a group as a whole. Apparently (b) achieves better system utilization.

where α is a balance factor, varying from 0 to 1. A value closer to 0 means favoring fairness more; that closer to 1 means favoring efficiency more. This value is preset can be adjusted dynamically.

Step 4: Sort the jobs with S_p.

Step 5: Group jobs with window size β, for each job window, do job allocation. The job allocation algorithm with window size β runs as follows: based on the permutation of the jobs, do greedy job allocation, and select one schedule with the least makespan, meaning that the jobs in the window generate a schedule with highest utilization rate.

4. ADAPTIVE POLICY TUNING

The metric-aware scheduling allows system owners to tune the scheduling policy leaning toward their interested metrics. However, the effectiveness of a scheduling policy is also dependent on the workload characteristic which varies from time to time. To mitigate the impact of workload change, we introduce adaptive policy tuning as a complement to metric-aware scheduling. Basically, we enable the scheduler dynamically change scheduling policy (by tuning the balance factor and allocation window size) based on the feedback of the monitored metrics. For example, if the monitored waiting time is below the expected value, the balance factor will be adjusted to a value to enhance the waiting time. This may impact the fairness metric. But when the fairness is not satisfied, the balance factor will adjusted again to waiting time more.

To determine a specifice adaptitve policy tuning scheme, we need to determine following consideration or parameters.

First, we need to be aware of what metrics we are interested to monitor and impact through the metrics balancer. For example, if we are interested in balancing effiency and fairness, we need to tune balance factor BF. If we are interested in balancing system utiltization, we need to tune window size W. These conclusions are testified by our later experiments. BF and W should be able to be tuned separately or together.

Second, we need to determine the what detailed metrices we should monitor. For example, if we know we want to balance effiency and fairness, we should monitor the job waiting times of the current queue or the average waiting time of jobs in recent period of time. If those value is more than a threshold, we will tune the balance factor to favor short jobs (i.e. BF closer to 0).

Third, we need to determine the thresholds of the monitored metrics at which the policy tuning should be triggered. The proper set threshold will not only determine the effectiveness of the policy tuning but also controls the aggressiveness of policy tuning. One example is to enlarge the window size when the average system utilization rate a short-term past (e.g., in past 1 hour) is below the average of a longer-term past (e.g., in past one day).

Fourth, we need to determine the granularity of tuning. The granularity can be very flexible. For example the BF can be tuned between the two numbers 1 and 0, or can be tuned every one tenth from 1 to 0. The window size also can be added by one every time needed, while the overhead of the job allocation is acceptable.

A specific policy tuning scheme is determined when proper parameters are determined in above four aspects. Theres is no uniform rule to determine the parameters. Proper arguments can be determined base on the priorities of individual machines and the workload characteristic. Historical metrics statistics may help to choose the proper value. And the simulation based on recent workload is also intructive. In next section, we provide an example configurations of policy tuning scheme and evaluate it in the experiment parts as a case study.

5. EXPERIMENT

In this section, we present a series of experiments to evaluate our proposed scheduling mechanism. We begin with introduction of the experiment, followed by discussion on evaluation metrics and then experimental results.

5.1 Experiment setup

We conducted simulation based experiments on two sets of separate workloads from two kinds of different machines. One workload is from Blue Gene/P (BG/P) system (named Intrepid) at Argonne National Laboratory the other is from the SP2 system at San Diego Supercomputing Center (SDSC). The former is a large-scale MPP system with 163,840 cores and the latter is cluster system with 338 cores. The ANL BG/P workload contains nearly 3000 jobs during the first ten days in June 2010. The SDSC workload contains the first 5000 job (during two months) in job log from corresponding log in Parallel Workload Archive [2].

We implement our scheduling schemes into an event-driven simulator with different configurations for each of the workloads. Specifically, for BG/P workload, we use contiguous job allocation with which only logically contiguous computing nodes can together serve a single job. This is because BG/P system uses 3D torus partition-based network to manage computing nodes. And for SP2 workload, we use contiguous job allocation as this type of system allows any set of jobs to be grouped together to serve a single job. Meanwhile, we use conservative backfilling for BG/P workload which is actually running in Cobalt resource manager on Intrepid machine and use EASY backfilling for SP2 workload since it is commonly used on many similar cluster systems.

For each workload, we conducted several experiments. First, we evaluate the effect of using different balance factors (BF) and window size (W). Next, we explore in more detail how BF and W impact the change of metrics. Finally, we evaluate the effectiveness of adaptive policy tuning including tuning BF and W both separately and together. Next, we will discuss the detailed evaluation metrics followed with experimental results.

5.2 Evaluation metrics

Folloing metrics were used in the performance evaluation.

- *Waiting time*: the time period between when the job is submitted and when it is started. In our experiments we measure two kinds of waiting time statistics. First, we measure the average waiting time among all the jobs to reflect the "efficiency" of a scheduling policy. Second, in real-time metric monitoring, we examine the instant aggregate waiting time of all the jobs in the current queue in order to measure the depth of the queue. A high aggregate waiting value (or a deeper queue) may be caused by either there are a large number of jobs waiting or there are some jobs enduring very long waiting or both. The average waiting time is an average number during a time period while the aggregate waiting time is an instant number.

- *Fairness.* To assess fairness, we assign a "fair star time" to each job at its submission. Any job started after its "fair start time " is considered to have been treated unfairly. The "fair start time" is calculated as follows: assuming there is no later arrival jobs, we conducted a simulation of scheduling under current scheduling policy and get when the job will be started. We count the number of unfair job to assess the overall fairness. This metric has been used in existing work [16].

- *System utilization rate.* This metric represent the ratio of the utilized (or delivered) node-hour compared to total available node-hour in the checked period of time. Usually it is refer to an average value over time. Sometimes when we refer to the instant system utilization rate we count it as the ratio of the number of busy nodes to the total number of nodes.

- *Loss of capacity (LoC).* It's a metric relevant to system utilization. A system incurs LoC when (i) it has jobs waiting in the queue to execute and (ii) it has sufficient idle nodes, but it still cannot execute those waiting jobs because of fragmentation. A scheduling event takes place whenever a new job arrives or an executing job terminates. Let us assume the system has N nodes and m scheduling events, which occurs when a new job arrives or a running job terminates, indicated by monotonically nondecreasing times t_i, for $i = 1 \dots m$. Let n_i be the number of nodes left idle between the scheduling event i and $i + 1$. Let δ_i be 1 if there are any jobs waiting in the queue after scheduling event i and at least one is smaller than the number of idle nodes n_i, and 0 otherwise. Then loss of capacity is defined as follows:

$$LoC = \frac{\sum_{i=1}^{m-1} n_i (t_{i+1} - t_i) \delta_i}{N \times (t_m - t_1)}. \quad (4)$$

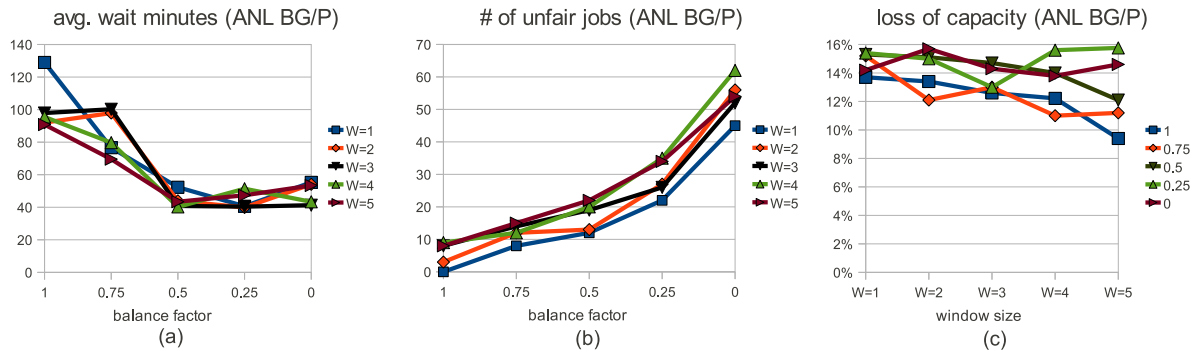


Figure 3: The effect of using balance factor and window size (ANL BG/P workload).

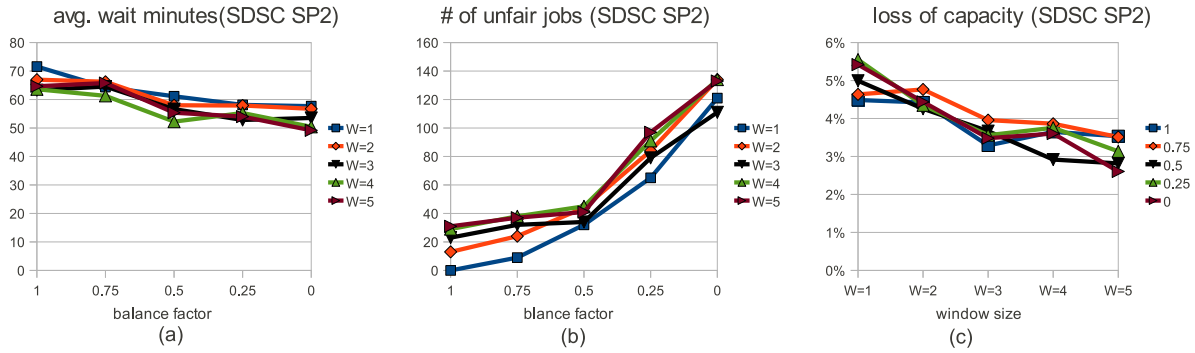


Figure 4: The effect of using balance factor and window size (SDSC SP2 workload).

Loss of capacity can reflect the resource fragmentation. In other word, the less of loss of capacity means the resources are better utilized. This metric has been used in our previous work [24] and a similar one has been used in related work [27].

5.3 Experimental results

5.3.1 *Static metric balancing*

In order to show the effect of metric-aware scheduling, we run simulations with various balance factor and job allocation window. Specifically, the balance factor is tuned between series $[1, 0.75, 0.5, 0.25, 0]$. A larger number represents a queuing policy closer to FCFS. Thus, BF being 1 represents perfect FCFS and BF being 0 represents perfect SJF. The size of job allocation window varies from 1 to 5.

Figure 3 shows the results of the BG/P workload. As shown in the Figure 3(a), the average waiting time declines as the balance factor decreases Especially, the waiting time drops significantly when BF tuned from 1 to 0.5, and kept at the same level when the BF continues decreases. This suggest that prioritizing jobs consider both job age (waiting time) and job length (expected running time) can enhance scheduling efficiency dramatically compared to pure FCFS (consider job age only). Theoretically, perfect short job first should lead to best average waiting time. But the results show that when BF tuned from 0.5 to 0, there is no clear improvement, which suggest that put too much weight on job length also bring some bad effect so that limits the further improvement on average waiting time. The window size does not show clear impact on average waiting time as the different lines are staggered along each other. But for FCFS, using a window size larger than 1 can achieve considerable improvement on average waiting time (by more than 20%) compared with traditional FCFS plus backfilling scheduling policy.

Figure 3(b) shows the fairness results of the BG/P workload. Clearly, the number of unfair jobs increases as the policy approaching FJS. And a general trend is seen that a larger window size also decreases fairness. Figure 3(c) shows the results of loss of capacity. Since this metric is influenced by window size more than balance factor, we put the window size to x-axis and the balance factor to legend in order to show the trends. We can see that when BF is no less than 0.5, the loss of capacity has the decreasing trend as the window size increases, meaning that enlarging job allocation window can help to utilize computing nodes more efficiently. This effect is not shown when the queuing policy getting closer to SJF (see the lines of BF=0.25 and BF=0).

Figure 4 shows similar results for SP2 workload. Generally, consistent trends are seen compared to BG/P workload. The average waiting are decreased consistently when the queuing policy getting closer to SJF (Figure 4(a)). And the window size provide a slight help in decrease average waiting (as shown in the figure window size 4 achieves best average waiting while windows size 1 has the worst). Figure 4(b) indicates that fairness declines when BF getting

smaller especially under 0.5. Figure 4(c) shows the loss of capacity decreasing as the window size getting larger.

5.3.2 *Adaptive policy tuning for efficiency/fairness*

Figure 5 shows the variation of the queue depth, measured by instant aggregate waiting time among all queuing jobs, over time. Figure 5(a) shows the data in normal scale and Figure 5(a) presents the logarithm scale on the same data sets helping to zoom the data characteristic under small scale.

Totally four lines are plotted in each figure. Three of them representing static metric-aware scheduling with BF=1, 0.75, and 0.5 respectively. We do not examine the cases with BF smaller than 0.5 here because they are attested being not good options in last set of experiments. Each line consists a series of plot on the time axis. Each plot represents the sum of the waiting times of jobs waiting in the queue at that instant of time. A high aggregate waiting value may be caused by either there are a large number of jobs waiting or there are some jobs enduring very long waiting or both. In later text, we call a queue with higher aggregate waiting value is "deeper" than the one with lower value. The value is checked every 30 minutes. Thus the plots are made every 0.5 hour on the x-axis, which is labeled with the elapsed hours from the time zero (the first job submitted).

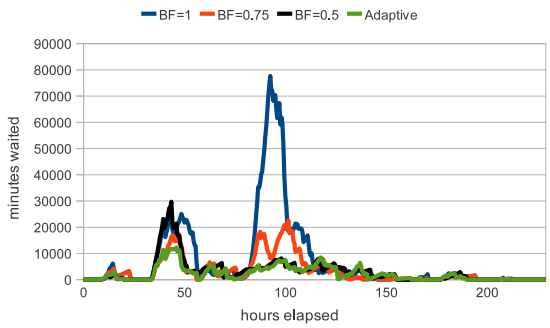
As shown in the figures, BF=1 is generally on the top of all other lines, meaning FCFS has the deepest queue, especially with a waiting job burst round 100th hours from the time zero. The policy with BF=0.75 does better than FCFS in dealing with the job burst; the maximum queue depth is only 1/4 of the FCFS one (around the 100th hour). The policy with BF=0.5 does even better; the maximum depth is only less than 1/8 of the FCFS one. However, when the queue is not deep, the FCFS does not bad, sometimes even better than the other two cases (see Figure 5(b) at elapsed hours around 150). This fact further encourage us to adaptively tune balance factor with the rationale that using a high BF when the queue is not deep and tuning the BF to a smaller value when the queue is deep.

The fourth line (legended by "adaptive") represents results from adaptive tuning the balance factor during runtime. Here the tuning point is when the aggregate waiting time reaches 10000 minutes. Specifically, when the aggregate waiting time is smaller than 10000 minutes, the BF is set to 1; otherwise, the BF is set to 0.5. The experimental results are surprisingly good that the overall queue depth of adaptive tuning is not only much better than FCFS, but also slightly better than the case with BF set to 0.5. This suggest that adaptive method takes advantage of different policies at corresponding queue status, which is desired by our original design.

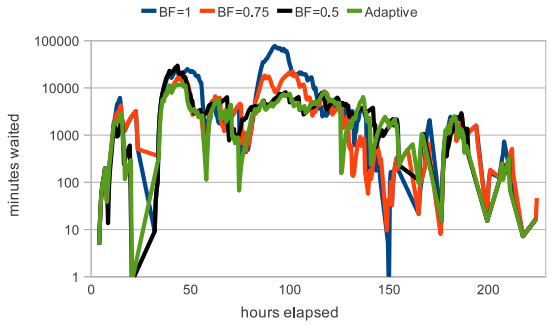
5.3.3 *Adaptive policy tuning for utilization*

This part of experimental results show the impact on system utilization rate by adjusting job allocation window size. We first examine the case that window size is not changed over time and then present the case with window size adaptively tuned. Figure 8 shows the results of the former case. By setting BF to 1 or 0.5, and window size to 1 or 4, we get four simulation cases, each shown in Figures 8(a) to (d).

Since for a workload, the average utilization rate is decided by the total node-hour of all jobs and the makespan, different simulations won't results in different utilization rate if the



(a)



(b)

Figure 5: Results of adaptively tuning balance factor.

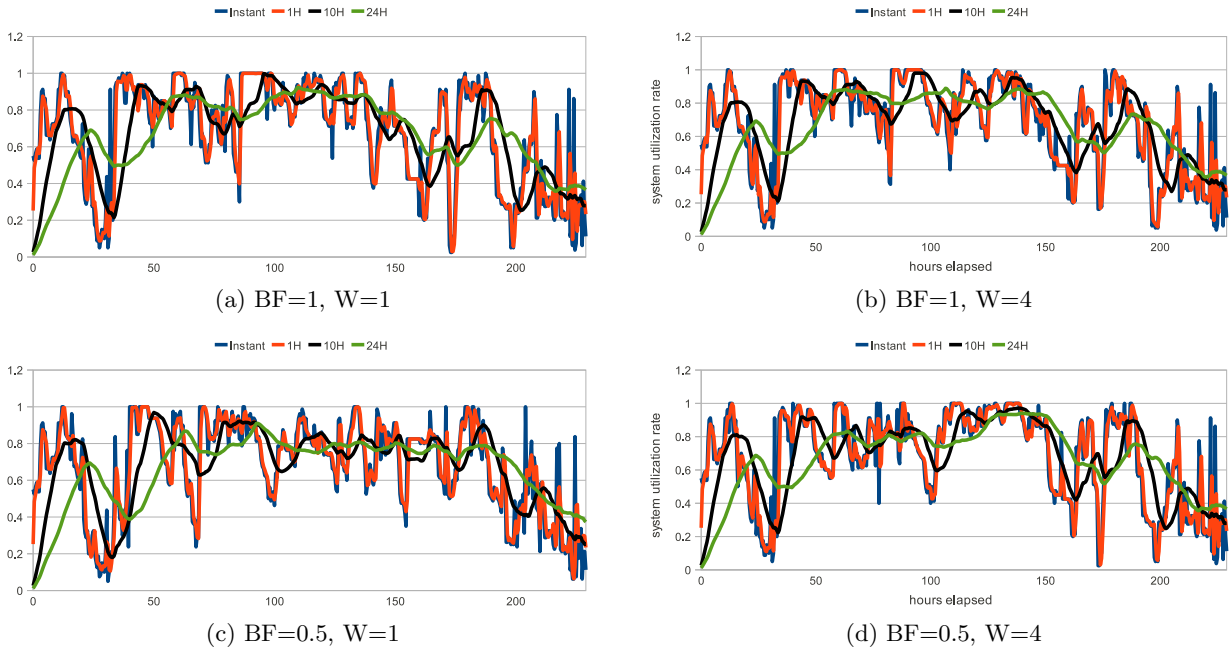
workload is not saturating the machine. Thus, in our experiment we monitor the instant utilization rate and the short-term averages. As in each subfigure, there are four plotted lines. A point on the "instant" line represents the instant system utilization rate. A point on the "1H" line represents the average system utilization rate during the past one hour. Similarly, the "10H" and "24H" the average in the past 10 hours and 24 hour, respectively. The values are checked by every 30 minutes. The x-axis represents the elapsed time in hour from time zero.

Comparing Figure 8(a) and Figure 8(b), we found the latter reduces the chances of the dipping the utilization, for example, at nearly 170th hour, the instant and 1H line in the former get closer to 0 but it is raised to 20% percent in the latter. This fact suggests that a larger window size help to mitigate the resource draining effect (shown in figure 2(a)) which hurt the system utilization.

Comparing Figure 8(c) and Figure 8(d),the main enhancement of the latter can be seen between the stable period (between 50th to 150th hour) which show a higher long-term average. That is, the 10H and 24H lines are generally higher in the latter than those in the former.

Now we add adaptive window size tuning and get the results in Figure 10. When BF is set to 1, we get a higher 24H line during the stable time compared to the cases without adaptive tuning. When BF is set to 0.5, the enhancement is not that clear shown in the figure. That may because BF set to 0.5 is already an efficient scheduling policy, leaving less room to improve for window size change.

5.3.4 *Two-dimensional policy tuning*



(a) BF=1, W=1

(b) BF=1, W=4

(c) BF=0.5, W=1

(d) BF=0.5, W=4

Figure 7: Monitoring of system utilization rate with different BF and W configurations.

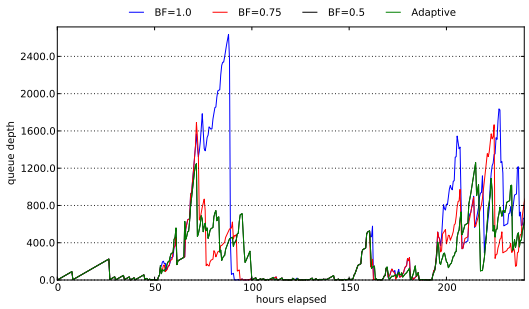


Figure 6: The effect on aggregate waiting of using 2D adaptive policy adjusting (SP2).

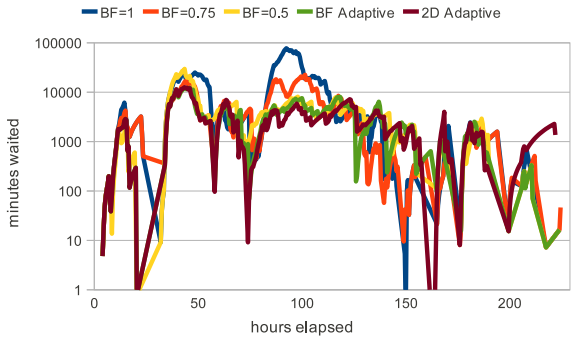


Figure 11: The effect on aggregate waiting of using 2D adaptive policy adjusting.

We already have experiments on tuning balance factor and window size respectively. Naturally we want to tune them together. We call this kind of tuning as two-dimensional policy tuning. It is the combination of the two tuning schemes presented earlier. Specifically, the balance factor and windows are both initialized to 1 and each of them follow their respective tuning strategy used in earlier experiments. We get the simulation results in Figure 11 and Figure 13.

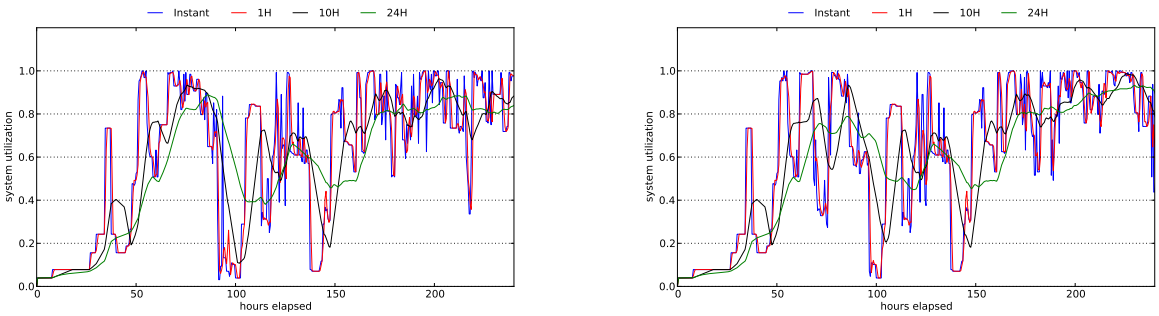
Figure 11 shows the queue depth change under 2D policy tuning compared to original strategies. We can see 2D adaptive tuning does even better than BF-only tuning. Not only it does good in avoiding queue depth busting, but also it performs very good when the queue is not deep. For example, it outperforms all other cases between hour 150 and 200.

Figure 13 shows the system utilization rate under 2D policy tuning. We observe that in this figure 10H and 24H line are more stable than those in previous figures. That is, system utilization are stabilized by 2D policy tuning. This indicates that 2D policy tuning help to achieve a stable system load, or in other words, a kind of temporal load balance, which will bring some potential benefit to achieve a sustainable resource management. For example, an evenly distributed system load may be good for efficient power management, causing less power waste by occasionally leaving substantial number of nodes idle.

6. RELATED WORK

7. SUMMARY AND FUTURE WORK

8. ACKNOWLEDGMENTS



(a) BF=1, W=1

(b) BF=1, W=4

(c) BF=0.5, W=1

(d) BF=0.5, W=4

Figure 8: Monitoring of system utilization rate with different BF and W configurations.

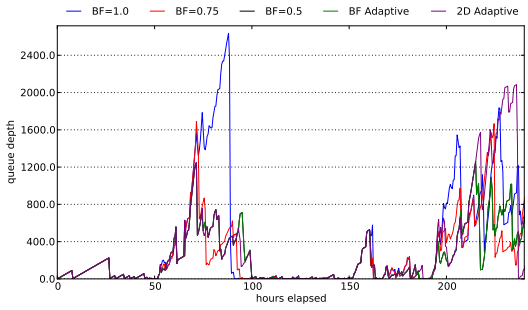


Figure 12: The effect on aggregate waiting of using 2D adaptive policy adjusting.

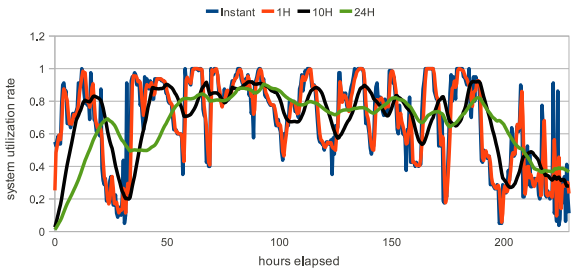


Figure 13: The effect on utilization rate of using 2D adaptive policy adjusting (BG/P).

This work is supported in part by National Science Foundation grants CNS-0720549, and CCF-0702737. The work at Argonne National Laboratory is supported by DOE Contract DE-AC02-06CH11357.

9. REFERENCES

[1] Cobalt project. *http://trac.mcs.anl.gov/projects/cobalt.*

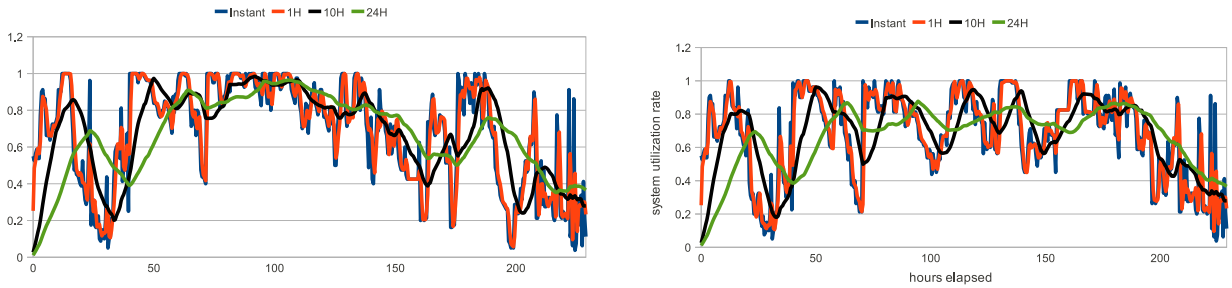
[2] Parallel workload archieve. *http://www.cs.huji.ac.il/labs/parallel/workload.*

[3] I. Ababneh and S. Bani-Mohammad. A new window-based job scheduling scheme for 2D mesh multicomputers. *Simulation Modelling Practice and Theory*, 19(1):482–493, 2011.

[4] N. Desai, R. Bradshaw, C. Lueninghoener, A. Cherry, S. Coghlan, and W. Scullin. Petascale system management experiences. In *Proc. USENIX Large Installation System Administration Conference (LISA)*, 2008.

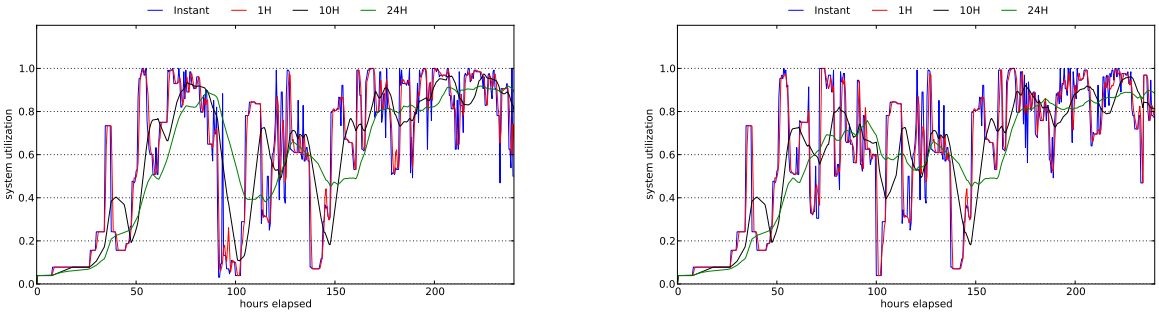
[5] D. Feitelson. Experimental analysis of the root causes of performance evaluation results: a backfilling case study. *IEEE Transactions on Parallel and Distributed Systems*, 16(2):175–182, 2005.

[6] S. Grothklags and A. Streit. On the comparison of cplex-computed job schedules with the self-tuning dynp



(a) BF=1 (b) BF=0.5

Figure 9: Effect on system utilization rate of adaptively tuning window size (BG/P)



(a) BF=1 (b) BF=0.5

Figure 10: Effect on system utilization rate of adaptively tuning window size (SP2)

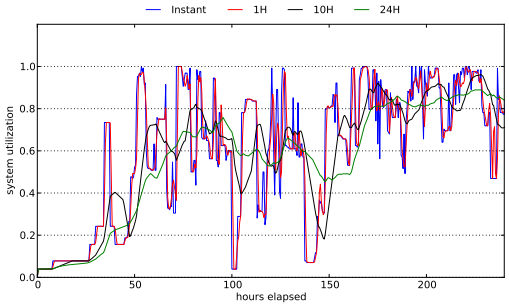


Figure 14: The effect on utilization rate of using 2D adaptive policy adjusting (SP2).

job scheduler. In *Proc. of IEEE International Parallel & Distributed Processing Symposium (IPDPS)*, 2004.

[7] S. M. Hussain Shah, K. Qureshi, and H. Rasheed. Optimal job packing, a backfill scheduling optimization for a cluster of workstations. *J. Supercomput.*, 54(3):381–399, 2010.

[8] D. Irwin, L. Grit, and J. Chase. Balancing risk and reward in a market-based task service. In *Proc. of IEEE International Symposium on High performance Distributed Computing (HPDC)*, 2004.

[9] D. Jackson, Q. Snell, and M. Clement. Core algorithms of the Maui scheduler. In *Job Scheduling Strategies for Parallel Processing, LNCS 2221*, pages 87–102, 2002.

[10] J. P. Jones and B. Nitzberg. Scheduling for parallel supercomputing: A historical perspective of achievable utilization. In *Proc. of the Job Scheduling Strategies for Parallel Processing (JSSPP)*, 1999.

[11] N. Kaushik, S. Figueira, and S. Chiappari. Flexible time-windows for advance reservation scheduling. In *Proc. of IEEE International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS)*, 2006.

[12] D. Krishnamurthy, M. Alemzadeh, and M. Moussavi. Towards automated HPC scheduler configuration tuning. *Concurrency and Computation: Practice and Experience*, 23(15):1723–1748, 2011.

[13] C. B. Lee and A. E. Snavely. Precise and realistic utility functions for user-centric performance analysis of schedulers. In *Proc. of International Symposium on High Performance Distributed Computing (HPDC)*, 2007.

[14] C. Lu, J. A. Stankovic, G. Tao, and S. H. Son. Design and evaluation of a feedback control EDF scheduling algorithm. In *Proc. of IEEE Real-Time Systems Symposium (RTSS)*, 1999.

[15] D. Raz, H. Levy, and B. Avi-Itzhak. A resource-allocation queueing fairness measure. In *Proc. of the joint international conference on Measurement and modeling of computer systems (SIGMETRICS)*, 2004.

[16] G. Sabin, G. Kochhar, and P. Sadayappan. Job fairness in non-preemptive job scheduling. In *Proc. of*

International Conference on Parallel Processing (ICPP), 2004.

[17] E. Shmueli and D. G. Feitelson. Backfilling with lookahead to optimize the packing of parallel jobs. *J. Parallel Distrib. Comput.*, 65(9):1090–1107, 2005.

[18] S. Srinivasan, R. Kettimuthu, V. Subramani, and P. Sadayappan. Selective reservation strategies for backfill job scheduling. In *Job Scheduling Strategies for Parallel Processing, LNCS 2357*, pages 55–71, 2002.

[19] A. Streit. A self-tuning job scheduler family with dynamic policy switching. In *Job Scheduling Strategies for Parallel Processing, LNCS 2357*, pages 1–23, 2002.

[20] A. Streit. Evaluation of an unfair decider mechanism for the self-tuning dynp job scheduler. In *Proc. of IEEE International Parallel & Distributed Processing Symposium (IPDPS)*, 2004.

[21] A. Streit. The self-tuning dynP job-scheduler. In *8th International Heterogenous Computing Workshop (HCW)*, 2004.

[22] W. Tang, N. Desai, D. Buettner, and Z. Lan. Analyzing and adjusting user runtime estimates to improve job scheduling on the Blue Gene/P. In *Proceedings of IEEE International Parallel & Distributed Processing Symposium (IPDPS)*, 2010.

[23] W. Tang, Z. Lan, N. Desai, and D. Buettner. Fault-aware, utility-based job scheduling on Blue Gene/P systems. In *IEEE International Conference on Cluster Computing*, 2009.

[24] W. Tang, Z. Lan, N. Desai, D. Buettner, and Y. Yu. Reducing fragmentation on torus-connected supercomputers. In *Proc. of IEEE International Parallel & Distributed Processing Symposium (IPDPS)*, 2011.

[25] W. A. Ward, C. L. Mahood, and J. E. West. Scheduling jobs on parallel systems using a relaxed backfill strategy. In *Job Scheduling Strategies for Parallel Processing, LNCS 2357*, pages 88–102, 2002.

[26] Y. Yuan, G. Yang, Y. Wu, and W. Zheng. PV-EASY: a strict fairness guaranteed and prediction enabled scheduler in parallel job scheduling. In *Proc. of the 19th ACM International Symposium on High Performance Distributed Computing (HPDC)*, 2010.

[27] Y. Zhang, H. Franke, J. Moreira, and A. Sivasubramaniam. Improving parallel job scheduling by combining gang scheduling and backfilling techniques. In *Proc. of IEEE International Parallel & Distributed Processing Symposium (IPDPS)*, 2000.