Adaptive Metric-Aware Job Scheduling for Production Supercomputers

Wei Tang,[†] Dongxu Ren,[*] Narayan Desai,[†] Zhiling Lan[*]

[†]Argonne National Laboratory, [*]Illinois Institute of Technology

Sep 10, 2012

Outline

Motivation

Job scheduler is an important component on supercomputers

- prioritizing queue for user satisfaction
- making efficient use of resources

Motivation

Job scheduler is an important component on supercomputers

- prioritizing queue for user satisfaction
- making efficient use of resources

Problem 1: scheduling goals are various

- Different goals from user and system owner
- Related but conflicting

Motivation

Job scheduler is an important component on supercomputers

- prioritizing queue for user satisfaction
- making efficient use of resources

Problem 1: scheduling goals are various

- Different goals from user and system owner
- Related but conflicting

Problem 2: workload characteristics are amorphous

- Effectiveness of a scheduling policy depends on workloads
- But, workload characteristics keep changing unpredictably

Motivation

Job scheduler is an important component on supercomputers

- prioritizing queue for user satisfaction
- making efficient use of resources

Problem 1: scheduling goals are various

- Different goals from user and system owner
- Related but conflicting

Problem 2: workload characteristics are amorphous

- Effectiveness of a scheduling policy depends on workloads
- But, workload characteristics keep changing unpredictably

Thus, it's hard to design a versatile scheduling policy

Solution Overview

Adaptive Metric-Aware Scheduling Framework

Solution Overview

Adaptive Metric-Aware Scheduling Framework

Metric-aware job scheduling

- balance different interests by metrics
- e.g., queuing effiency, fairness, system utilization and cost

Solution Overview

Adaptive Metric-Aware Scheduling Framework

Metric-aware job scheduling

- balance different interests by metrics
- e.g., queuing effiency, fairness, system utilization and cost

Adaptive policy tuning

- dynamically tune scheduling policy based on feedback
- mitigate the impact of varying workload characteristics

Solution Overview

Adaptive Metric-Aware Scheduling Framework

Metric-aware job scheduling

- balance different interests by metrics
- e.g., queuing effiency, fairness, system utilization and cost

Adaptive policy tuning

- dynamically tune scheduling policy based on feedback
- mitigate the impact of varying workload characteristics

Provide a *balanced* and *sustainable* scheduling mechanism

Diagram of our solution

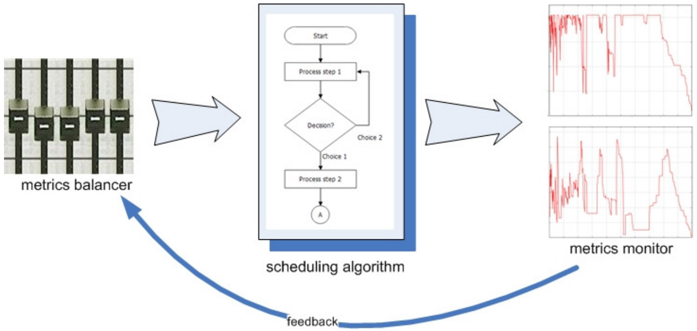


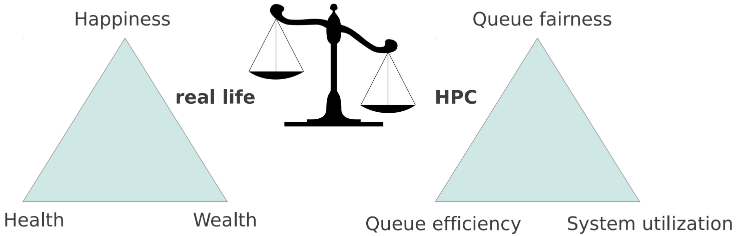
Figure : Diagram of adaptive metric-aware job scheduling framework.

Metric overview

- Quantified criteria
- Reflecting certain interest from either user or system
- User satisfaction
    - job waiting time
    - slowdown
    - fairness
    - etc
- System perspective
    - system utilization rate
    - resource fragmentation
    - power efficiency
    - etc

To be balanced

Balance is needed everywhere!



What to balance

METRICS TO BE BALANCED

- **Queuing efficiency**
  - regarding the time of job waiting
  - avg. job waiting, response time, slowdown, etc
- **Queuing fairness**
  - no later-arrival jobs should delay early ones
  - psychologically, fairness is more important than efficiency
- **System utilization**
  - make efficient use of resources, minimizing wasted core-hours
  - system utilization rate, loss of capacity

Flaws of existing ways of scheduling

- **FCFS** (first come, first served)
  - good for fairness
  - bad for job waiting
  - prone to fragmentation
- **SJF** (short job first)
  - minimizing average waiting
  - bad for fairness
  - prone to starvation
- **MXF** (maximum x-factor first)
  - prioritizing by $waittime/runtime$
  - act in between FCFS and SJF
  - cannot balance at will
- **Job allocation scheme**
  - allocate jobs one by one in queue order
  - job allocation loses flexibility after jobs sorting

Our approach to balance

- **Balance factor (BF) in job sorting**
  - BF tunable from 0 to 1.
  - tune queuing policy between FCFS (BF=1) and SJF (BF=0)
  - balance between fairness and efficiency

Our approach to balance

- **Balance factor (BF) in job sorting**
    - BF tunable from 0 to 1.
    - tune queuing policy between FCFS (BF=1) and SJF (BF=0)
    - balance between fairness and efficiency
- **Window based job allocation.**
    - after sorting, group jobs by window size W ($W \geqslant 1$)
    - jobs within the same window can be allocated as a whole (no priority difference)
    - a larger window provides more flexibility to pack jobs

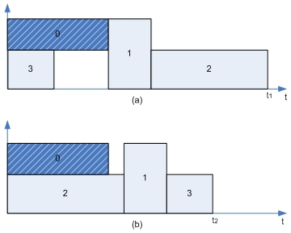


Figure : An example showing the limitation of allocating jobs one by one. (a) one-by-one in queue order; (b) as a whole (W=3)

Scheduling Algorithm

- **Step 1**: calculate waiting score for job i, mapping to [0,100]
  - $S_w = 100 \times \frac{wait_i}{wait_{max}}$
- **Step 2**: calculate walltime score for job i, mapping to [0,100]
  - $S_r = 100 \times \frac{walltime_{max} - walltime_i}{walltime_{max} - walltime_{min}}$
- **Step 3**: calculate balanced priority score
  - $S_p = BF \times S_w + (1 - BF) \times S_r$
- **Step 4**: sort all jobs by their balanced priority S_p
- **Step 5**: group jobs with window size W, for each window try job allocation. Select one schedule with minimum makespan.
- **Step 6**: make another pass to backfill remaining jobs

Adaptive policy tuning

- **Why adaptive tuning**
  - scheduling policy depends on workload characteristics
  - to counter the impact of workload variation
- **Existing ways addressing workload variation**
  - event-driven simulation on historical data (offline method)
  - or just ignore... (unfortunately this dominates)
- **Our proposed tuning scheme**
  - monitor interested metrics at runtime
  - adjust arguments of scheduling policies based on feedback
  - periodically check and adjust (e.g. every 30 minutes)

Parameters

- To configure a scheme for adaptive policy tuning, several parameters should be determined
  - what to tune, when to tune, how much to tune, etc

Table : Parameters to configure an adaptive scheme

Para.	Description	Possible values
T	tunable	BF or W
T_i	initial value of tunable	1 for both BF and W
Δ	the incremental value to tune T	0.5 for BF or 1 for W
M	monitored metrics	queue status or sys. util.
TH	threshold of M	(historical statistics)
E_p	event triggering T plus Δ	M reaches TH
E_m	event triggering T minus Δ	M reaches TH reversely
C_i	interval between check points	30 minutes

Algorithm

Algorithm 1: adaptive scheduling

```
$T = T_i$;                                    // initialize the tunable
while $True$ do
    if $now - last\_checked > C_i$ then           // at check point
        $m$ = get_monitored_values();           // get values of $M$
        $e$ = check_event($m$);      // compare feedback with $TH$
        if $e == E_p$ then
            $T = T + \Delta$ ;             // increase tunable by $\Delta$
        end
        if $e == E_m$ then
            $T = T - \Delta$ ;             // decrease tunable by $\Delta$
        end
        $last\_checked = now$ ;          // reset check point clock
    end
    schedule_jobs($T$) ;               // do real scheduling stuff
    sleep($SchedInterval$) ;          // sleep for several seconds
end
```

Outline

- Motivation
- Solutions
- **Experiments**
- Summary & Future Work

Experiment setup

- Cobalt resource management system
  - http://trac.mcs.anl.gov/projects/cobalt/
- Simulation based evaluation (Qsim)
- Real workload from production BG/P at ANL
- 163,840 cores, 9300 jobs

Metrics

- **Average waiting time**
  - time between job submission and job start (all job average)
- **Queue depth**
  - the sum of waiting times of all current queuing jobs
  - high queue depth means either a large number of waiting jobs or some jobs enduring long wait or both
- **Unfair jobs**
  - the number of jobs delayed by later arrival jobs
- **Utilization rate**
  - the ratio of delivered core-hours to total core-hours
- **Loss of capacity**
  - the ratio of idle core-hours while there are jobs waiting to the total core-hour
  - wasted system utilization (by fragmentation)

Metrics balance with balance factor and window size

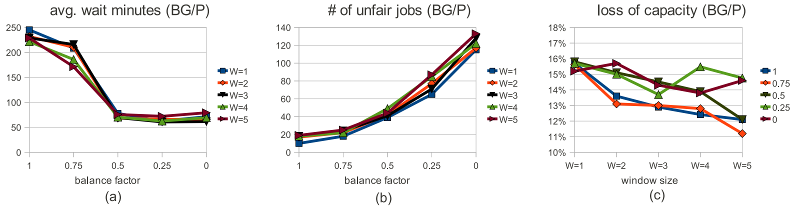
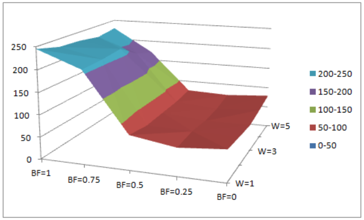
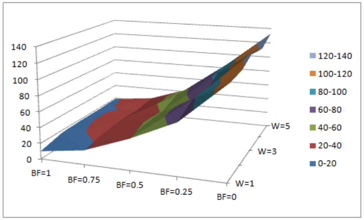


Figure : The effect of using balance factor and window size (BG/P)

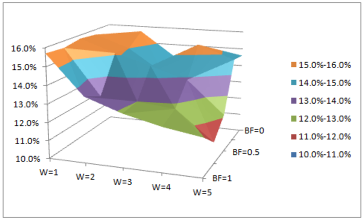
Metrics balance with balance factor and window size



(a) avg. wait



(b) unfair job



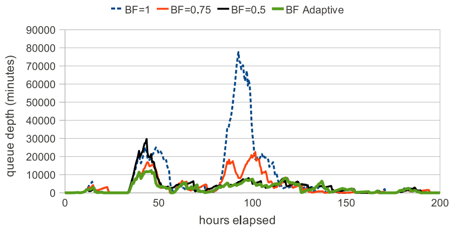
(c) loss of capacity

Configuration for adaptive scheduling

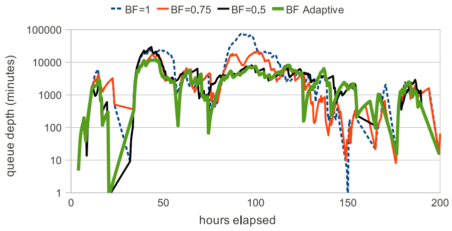
T	BF	W
T_i	1	1
Δ	0.5	4
M	queue depth (Q)	system utilization rate
TH	$\delta = Q - Avg(1m)$	$\delta = Avg(10h) - Avg(24h)$
E_p	$\delta_{i-1} > 0$ & $\delta_i < 0$	$\delta_{i-1} < 0$ & $\delta_i > 0$
E_m	$\delta_{i-1} < 0$ & $\delta_i > 0$	$\delta_{i-1} > 0$ & $\delta_i < 0$
C_i	30 minutes	30 minutes

- Avg(X) means the average value during last X period of time, e.g. 10 hours, 24 hours, 1 month.
- δ_i and δ_{i-1} means the checked value at current and last check point, respectively.

Queue depth influenced by tuning balance factor (BG/P)

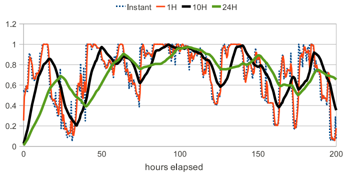


(d) queue depth

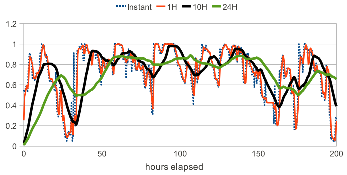


(e) queue depth (logarithm scale)

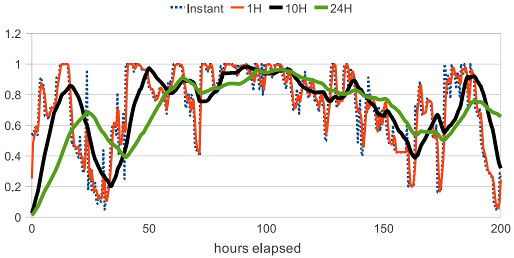
Monitoring of system utilization rate (BG/P)



(a) W=1



(b) W=4

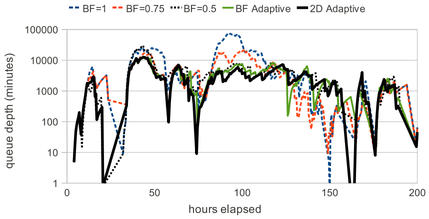


(c) W=Adaptive

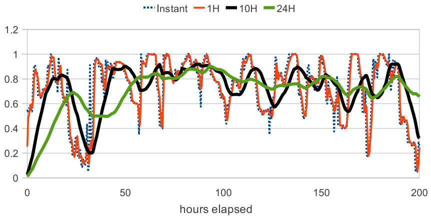
2D adaptive tuning (BG/P)

2D ADAPTIVE TUNING

- tune both BF and W simualtanously
- each follows respective configuration
- influential to both queue depth and system utilization



(a) queue depth



(b) system utilization rate

Overall improvement (BG/P)

Table : Improvement of adaptive tuning (BG/P)

configuration	avg. wait (min)	unfair #	LoC (%)
BF=1/W=1	245.2	10	15.7
BF=1/W=4	221.6	18	12.4
BF=0.5/W=1	77.9	39	15.8
BF=0.5/W=4	70.4	49	13.9
BF Adapt.	74.1	21	12.8
W Adapt.	198.1	16	11.9
2D Adapt.	71.3	19	12.1

Compared with baseline, 2D Adapt saves avg. wait by 71%, reduces LoC by 23%, and doubles unfair jobs (much less than the case (BF=0.5/W=4) with comparable improvement).

Performance of scheduler

Table : Runtime per scheduling iteration (sec)

window size	executing time
W=1	0.021
W=2	0.034
W=3	0.069
W=4	0.117
W=5	0.584

The scheduling iteration is triggered about every 10 seconds in real systems (e.g. in Cobalt), thus a scheduling iteration less than 1 second is affordable.

Summary

- Proposed adaptive metric-aware job scheduling
  - metric-aware job scheduling to balance competing objectives
  - adaptive policy tuning to counter the impact of varying workload characteristics
- Conducted simulation-based experiments
  - tested real workloads from multiple supercomputing centers
  - examined a variety of metrics such as job waiting time, queue depth, fairness, system utilization rate, and loss of capacity
  - demonstrated our scheduling methods improve system performance in a balanced and sustainable fashion

Future work

- Optimize window-based job allocation algorithm
  - to support larger window with limited overhead
  - consider distributed algorithms
- Employ feedback-control theory
  - to consolidate the adaptive policy tuning
- Expand the spectrum of metrics to be balanced
  - especially for systems cost such as energy consumption, system reliability, etc

Thanks you!