Efficient Evaluation of Scheduling Metrics Using Emulation: A Case Study in the Effect of Artefacts

Claudio Barberato*, Peter E. Strazdins*, Eric McCreath*, Muhammad Atif†

*:Computer Systems Group,
Research School of Computer Science,
The Australian National University
†: National Computational Infrastructure, Canberra

(slides available from http://cs.anu.edu.au/~Peter.Strazdins/seminars)

SRMPDS2018: The 14th International Workshop on Scheduling and Resource Management for Parallel and Distributed Systems. Eugene, OR
13 August 2018

THE AUSTRALIAN NATIONAL UNIVERSITY

1 Talk Overview

- motivation: why are scheduling algorithms important, how do we evaluate them? (metrics)
- background: resource management systems and scheduling
- how SLURM cluster configurations can support emulation
- infrastructure for emulation - the Sleep Program
- classical metric artefacts evaluation:
  - experimental setup and methodology
  - artefacts: non-determinism and submission order
- using emulation to reliably compare the (prioritized) Suspend/Resume vs Backfill scheduling algorithms
- improving emulation performance
  - trace sampling (balanced but too sensitive)
  - time shrinking, and its limitations
- conclusions and future work

2 Motivation: Why are Scheduling Algorithms Important?

- have a significant impact in the optimal utilization of HPC facilities
- classical metrics include overall system utilization (UT), and, averaged over all jobs, waiting time (WT), response time (RT), slowdown (SD) and weighted (by the number of cores) slowdown (WS)
- the dominant characteristic of such algorithms in practice is space-sharing
- of these, FCFS with Backfill-based algorithms have become dominant, with very few alternatives explored in recent literature or deployed in practice
  - an exception was Suspend/Resume, which was implemented into a customized version of PBS at ANU over the years 2005—2013
  - this achieved very high utilization when very large parallel jobs were submitted (and was reputed to have excellent UT at other times)
- the metrics used to compare scheduling algorithms are however subject to the effects of *artefacts*, which may render conclusions from many comparisons meaningless

3 Motivation: How are Scheduling Algorithms Evaluated?

- on a real supercomputer, various algorithms and their associated policies/parameters can be evaluated over a long period
  - time consuming and out of reach of ordinary researchers
- use of a simulator, which abstracts real jobs to submission time, estimated/actual runtime and memory usage, number of cores requested
  - typically use job traces derived from standard supercomputer workloads
  - very efficient and the results are highly reproducible
  - ignores real-world effects such as non-determinism in the network and delays introduced by the job scheduler
  - there is no guarantee that the scheduling algorithm being studied, e.g. FCFS with EASY Backfill behaves the same way in the real job scheduler and in the simulator

4 Motivation: How are Scheduling Algorithms Evaluated? (II)

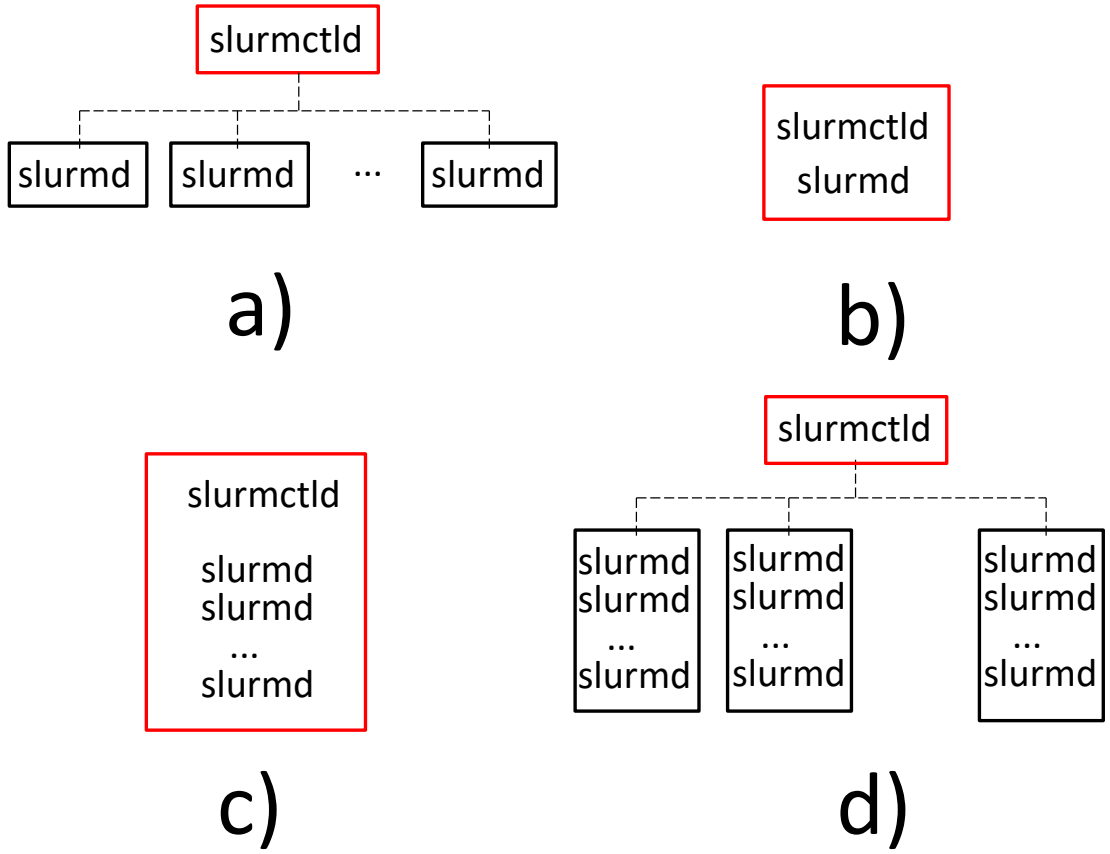
- the emulation approach:
  - again uses job traces, but uses 'sleep' programs instead of real ones and submit these jobs to a real scheduler
    - thus possible to emulate a far bigger system than is actually used
  - overcomes the deficiencies of simulations
  - problems: potentially very long waiting periods, and unclear whether it can scale to very large systems

5 Background: Resource Management Systems and Scheduling

- RMS administrate the utilization of supercomputer resources, such as CPU, memory, storage, and network
- one of their most important tasks is the scheduling of jobs submitted by users
- for this reason, they are often referred to as job schedulers, with PBS, MAUI, Torque and SLURM being popular RMS's with highly tuned schedulers
  - due to its high degree of configurability, we choose SLURM (Simple Linux User-level Resource Manager) as the basis of our emulation framework (of course, SLURM is no longer simple. . .)
  - it manages the access to computing resources, and it schedules (starts, executes and monitors) jobs
  - its two main components are a slurmd, a daemon running on each compute node which controls jobs on that node, and a slurmctld, running on the head node, which has the job scheduler

6 SLURM Cluster Configurations Supporting Emulation

- (a) SLURM supporting a real cluster
- (b) Front-end mode: the slurmd is on the head node
- (c) as for (b), but multiple slurmds to emulate a cluster
- (d) emulating a larger cluster, using multiple nodes, with the slurmctld isolated on the head node



- to emulate a supercomputer, we use (d) with 10 real nodes with 360 slurmds each, with each emulated node configured to have 16 cores (57600 cores in total, matching NCI's supercomputer Raijin)

7 Emulation Software – Sleep Program

- it is simple to write a program that sleeps for a specified time, which appears to be sufficient to support emulation
  - very small memory & CPU footprint; hence many sleep programs may be running via multiple slurmds on a single physical node
- the `nanosleep()` program had to be used to ensure an accurate sleep period was obtained
- to support suspend/resume, the `SIGTSTP` and `SIGCONT` signals, used by slurmd to suspend & resume a job respectively, had to be trapped, and the interval in-between must not be counted to the sleep time
- when the sleep job terminates, it writes in a common log file all pertinent information about its run (submission/start/stop/suspend time, # cores requested) (the classical metrics are evaluated using these)
- a global batch file uses the wall clock time to submit jobs to the slurmctld faithfully according to the trace being used

8 Classical Metric Artefacts Evaluation: Experimental Setup and Methodology

- used a workload containing some 4M jobs submitted to NCI's supercomputer between April–November 2016
  - the vast majority of these jobs only required a single core, and only occasionally a very large highly parallel jobs was submitted
- we used a workload file which had for each job its priority, submission delay, number of CPUs, and expected & actual runtime
- a certain number of jobs are used to 'warm-up' the system; the remaining jobs are submitted according to their submission times and are used to evaluate the metrics
- both head and compute nodes were VMs with 2 real cores allocated to each

9 Classical Metric Artefacts Evaluation: Non-determinism

- to illustrate the effects of non-determinism even in an emulated system, a 1K job workload on a 3K node CPU cluster was run 5 times consecutively. Times (AVE, SSD) are in seconds, RSD is a percentage:

Run	WT	RT	SD	WS	UT
AVE	1281	1409	66.5	109.5	91.2
SSD	48.9	18.9	1.4	4.1	0.7
RSD	0.4	1.3	2.0	3.8	0.8

- even this simple experiment shows that the inherent non-determinism in real, distributed systems is significant, especially for the slowdown metrics
- these effects will not be demonstrated by any (simplistic) simulations of scheduling algorithms

10 Classical Metric Artefacts Evaluation: Submission Order

- the literature has observed that workloads from different HPC facilities can have a profound effect and scheduler performance, even between EASY and Conservative Backfill
- we used a slightly larger subset of the NCI trace, and ran it 10 times with random shuffling of job submission order using a Backfill scheduler:

Run	WT	RT	SD	WS	UT
AVE	1416	1544	73.4	113.2	85.2
SSD	205	205	11.3	19.3	2.9
RSD	14.5	13.3	15.4	17.0	3.4

- we now see from the RSD, which with the possible exception of utilization, shows differences that are likely greater than different scheduling algorithms would produce

11 Classical Metric Comparison: Prioritized Suspend/Resume vs Backfill

- in general, the introduction of priorities worsens all metrics, due to putting more constraints on the scheduler (more details in the paper)
  - note however that Suspend/Resume relies on priorities to handle large parallel jobs well, and also prevent the starvation of jobs victimized for suspension
- we use the same workload as on the previous slide, again shuffled 10 times to average out the effect of artefacts
  - 'Random Priorities' means 20% of randomly chosen jobs get high priority, and the remainder have low
  - 'Size-based Priorities' means only jobs requesting $\geq$ 700 CPUs have high priority
  - the submission order was kept the same, regardless of priority

12 Classical Metric Comparison: Suspend/Resume vs Backfill (II)

	Backfill: Random Priorities					Backfill: Size-based Priorities				
	WT	RT	SD	WS	UT	WT	RT	SD	WS	UT
AVE	1450	1578	71.5	143.9	86.7	1976	2104	83.1	102.4	87.6
SSD	217	217	9.0	13.9	3.4	400	400	18.5	21.3	3.7
RSD	14.9	13.7	12.6	9.7	3.9	20.2	19.0	22.3	20.8	4.2
	S/R: Random Priorities					S/R: Size Based Priorities				
	WT	RT	SD	WS	UT	WT	RT	SD	WS	UT
AVE	1699	1910	80.0	128.8	87.8	2470	2649	109.0	111.1	87.6
SSD	212	201	9.9	15.6	5.1	240	233	12.2	16.5	3.7
RSD	12.5	10.5	12.4	12.1	5.8	9.7	8.8	11.2	14.9	4.2

We can make some clear conclusions on this workload:

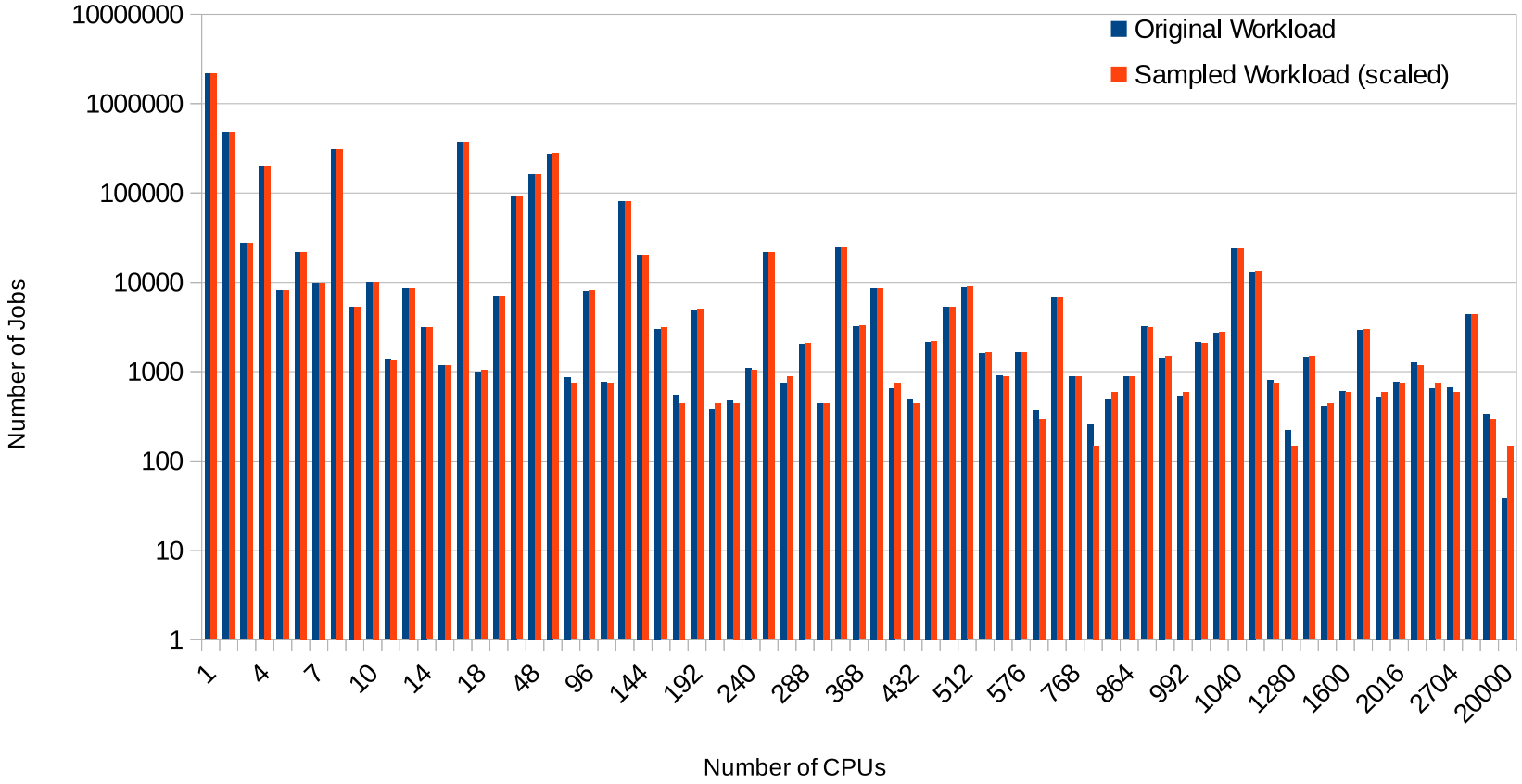
- $Backfill < Backfill(rand) < SR(rand) < Backfill(size) < SR(size)$
- $Backfill(rand) < Backfill < SR(rand) < Backfill(size) < SR(size)$
- $Backfill(size) < SR(size) < Backfill < SR(rand) < Backfill(rand)$

13 Improving Emulation Performance: Trace Sampling

- real traces from a supercomputer lasting 6 months or so are impractical for emulation
- sampling, or taking a representative subset, offers an attractive solution
  - neither random selection nor choosing small time sub-intervals are effective
  - the following procedure however yields better results:
    1. successively order the original workload by the required runtime, runtime and size of each job
    2. Select every job j_i in the ordered workload such that $j_i = \lfloor s\, i \rfloor + o$ where s is the chosen sampling ratio and $0 \leq o < s$ is an arbitrary offset
    3. re-order the selected jobs according to their submission order in the original workload

14 Improving Emulation Performance: Trace Sampling (II)

This sampling with a factor of $r = 147.2$ indicates the above procedure preserves the proportion of the sizes of the jobs, in terms of CPUs:



15 Sensitivity of the Balanced Trace Sampling Procedure

- the selection of the offset $0 \leq o < s$ is arbitrary and should not have a significant effect on the metrics
- the following are the classical metrics for 10 different workloads with a sampling ratio of $r = 147.2$, each with different offsets, using Backfill

	WT	RT	SD	WS	UT
AVE	1284	1881	179.9	77.9	97.0
SSD	381	151	53.3	21.4	0.5
RSD	29.6	8.0	29.6	27.5	0.5

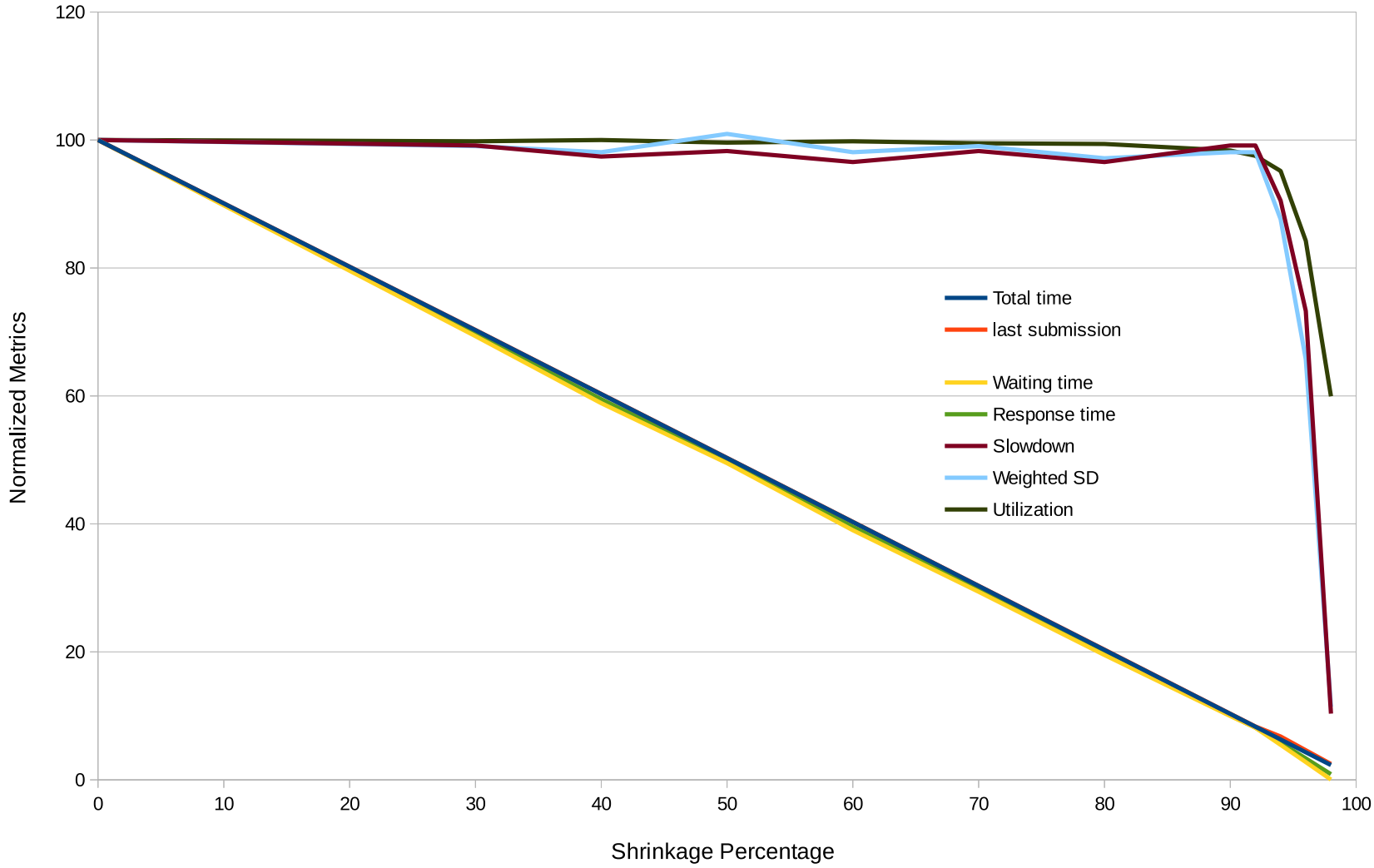
  The RSD values are very high, higher than for shuffling (though as before UT remains stable)

- it remains an open question whether:
  - a sampling procedure can be derived that preserves the job size and length distribution but is not sensitive to arbitrary parameters
  - what is the relationship of r to this sensitivity / inaccuracy?

16 Improving Emulation Performance: Time Shrinking

- another way to improve emulation efficiency is simply to decrease the submission, expected and actual run times of a workload by a shrinking factor $s \geq 1$
  - we expect this would decrease linearly the time-based metrics WT and RT; but 'normalized' metrics like SD, WS and UT should remain constant
  - we found that there was a time delay between the ending of a job and the starting of the next; this was constant and did not decrease with s. On our system, this drift was ≈ 0.1s. Adding a constant time of the same value to consecutively to each submission time significantly improved the range of s before the metrics broke down
- the results on the following page use drift correction for a 30,000 jobs processed in a 57600-CPU cluster show $s \leq 9.4$ preserves the metrics of the original ($s = 1$) workload

17 Improving Emulation Performance: Time Shrinking (II)



18 Conclusions

- simulation cannot capture real-world effects (network, scheduler, etc) that cause significant non-determinism
- artefacts, particularly job submission order, can dwarf quoted differences in scheduling algorithm performance
  - comparisons should use a series of shuffled workloads in order to be meaningful!
- emulation: tells you about the algorithm as implemented on the RMS!
  - SLURM's configurability makes it suitable to emulate large clusters
  - the Sleep Program needs special care to support Suspend/Resume
  - the batch submission script must carefully use wall time for good results
  - by taking into account the artefacts, we could reliably compare Backfill and Suspend/Resume with various priority scenarios
- some metrics (SD, WS) seem inherently unstable – should they be discarded?

19 Conclusions (II) and Future Work

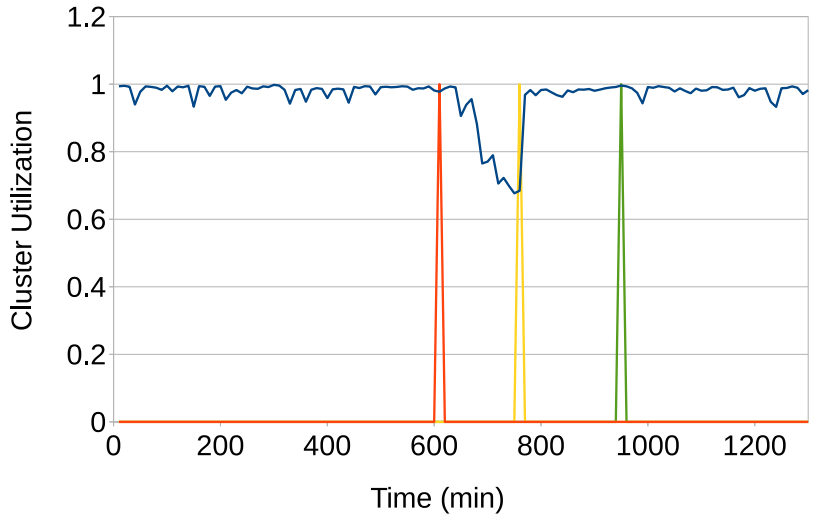
- improving emulation performance
  - sampling is needed, but it is hard to find a representative sample that is not over-sensitive to arbitrary choices
  - time shrinking can improve performance up to a factor of 9.4 without distorting metrics
  - the fundamental reason is that scheduler delays become increasingly significant
  - the speedups are still comparable to simulation-mode versions of SLURM (which are problematic to maintain)
- future work
  - a more thorough and systematic investigation of the use of emulation
    - more systematic evaluation of sampling, with a range of factors
    - a more thorough quantification of the effect of artefacts
    - requested and actual memory considered (especially for S/R)
  - review comparisons of scheduling algorithms in the literature

Thank You!! ...Questions???

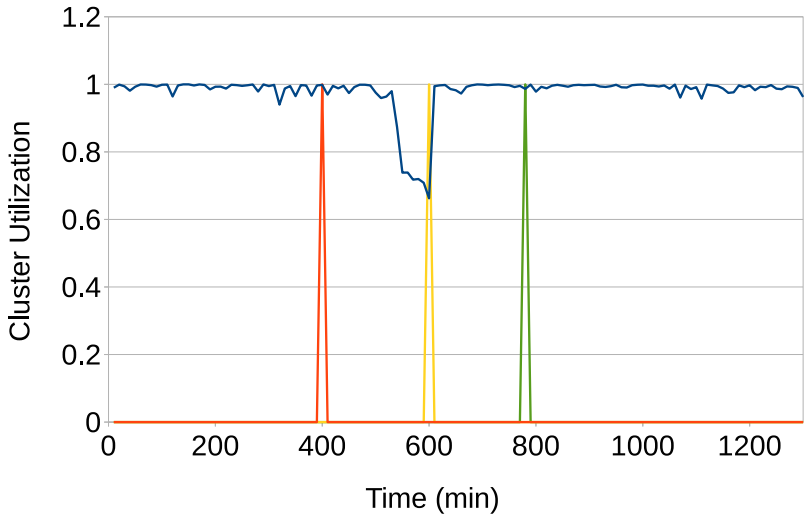
(email peter at cs.anu.edu.au)

20 Appendix: Backfill and Suspend/Resume Evaluation

- as a case study, we use our emulation framework to compare the standard SLURM Backfill algorithm, and Suspend/Resume
- use two modes: Linear: jobs must request whole nodes, and Consumable Resources: a job can request individual CPUs within a node
- the following shows Backfill, with a large parallel job submitted at a certain time (the red spike); the blue line represents utilization



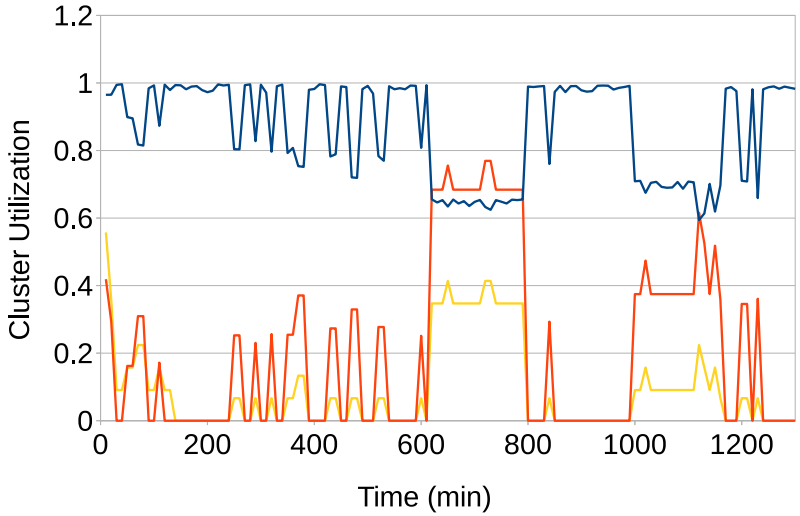
(a) Consumable Resources plug-in



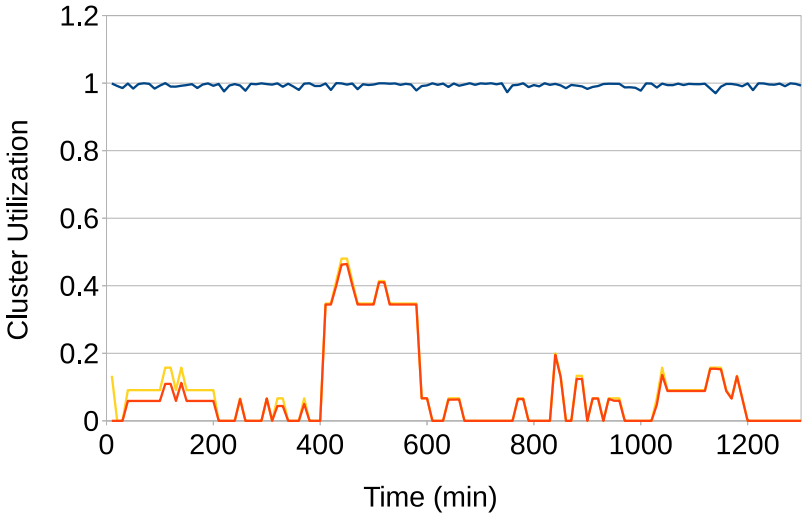
(b) Linear plug-in

21 Appendix: Backfill and Suspend/Resume Evaluation (II)

- as expected for Backfill, there is a loss of utilization from when the parallel job is scheduled and begins, waiting for enough cores to be freed
- with Suspend/Resume, the red (yellow) line shows the number of suspended low (active high) priority jobs (weighted by the number of cores



(a) Consumable Resources plug-in



(b) Linear plug-in

- we see highly sub-optimal behaviour for case (a), with approximately twice the number of jobs suspended as is necessary

22 Appendix: Backfill and Suspend/Resume Evaluation (III)

- Backfill and Suspend/Resume comparison using the Linear plugin (std-dev over 5 runs), showing the latter's utilization advantage in this scenario

Algorithm	WT	RT	SD	WS	UT
Backfill	718.2 (93)	123.8 (12)	291.7 (51)	135.2 (16.0)	97.7 (0.4)
S/R	567.0 (85)	119.9 (12)	227.2 (41)	105.1 (11.6)	99.3 (0.4)

- lesson for using Suspend/Resume in SLURM: either fix the Consumable Resources implementation or use the Linear plugin