Speculative Scheduling for Stochastic HPC Applications

Ana Gainaru[1], Guillaume Pallez (Aupy)[2], Hongyang Sun[1]
Padma Raghavan[1]

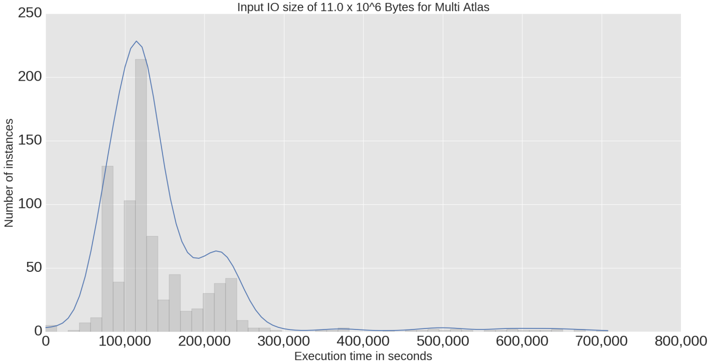
1. Vanderbilt University; 2. Inria & Univ Bordeaux

VANDERBILT UNIVERSITY

Scheduling Workshop, Bordeaux, June 27, 2019

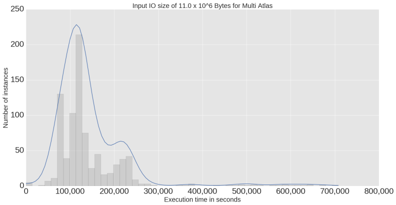
Stochastic HPC applications

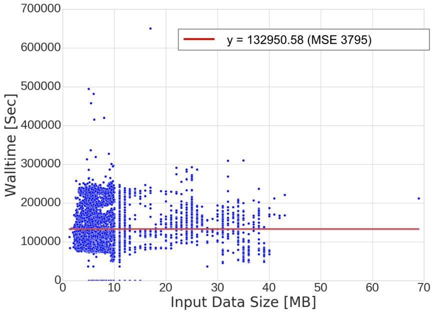
“second generation” HPC applications with heterogeneous, dynamic and data-intensive properties



Stochastic HPC applications

"second generation" HPC applications with heterogeneous, dynamic and data-intensive properties

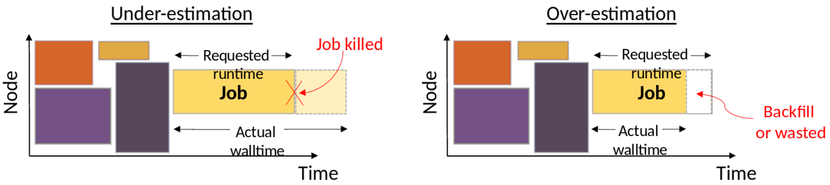




HPC current state

Reservation-based batch schedulers

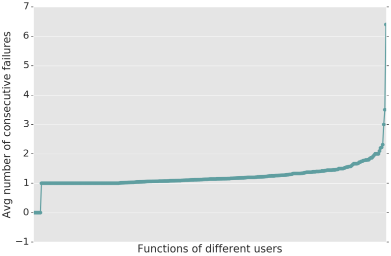
- Relies on (reasonably) accurate runtime estimation from the user/application



- Why not ask for more time than required?
- Why not trigger a schedule after every job end?

Backfilling algorithms

Over-estimated submissions	82.2 %
Under-estimated submissions	17.7%
Average over-estimation time	1.36 hours
Average over-estimation space	2132 node hours
Average small jobs size	48.6 nodes / 31 node hours
Percentage of small jobs	30.8%

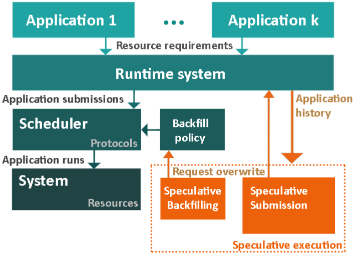


Intrepid (2009 ANL system)

Analyzing neuroscience workflows:
>30% of the submissions fail
1.25 hours of over-estimation space

Backfill performance depends on accurate estimates

Contribution



- Optimal sequence of requested times in the presence of small jobs that can be used for backfill
  - Overwrite the requested time at submission
- Speculative backfilling
  - Overwrite the request time temporarily during backfill

Model

- A system with P identical processors
- $\mathcal{J} = \{J_1, J_2, \ldots, J_M\}$ of **large** stochastic jobs
  - processor allocation p_j
  - walltime following different distributions
- A stream $\mathcal{B}$ of **small** jobs
  - arrival rate λ
  - an average execution time ε much smaller than that of the large jobs.

Continuous approximation

A stream of work arrives continuously in the queue with a rate $\mathrm{Z} = \lambda\varepsilon$

Model

- A system with P identical processors
- $\mathcal{J} = \{J_1, J_2, \ldots, J_M\}$ of **large** stochastic jobs
  - processor allocation p_j
  - walltime following different distributions
- A stream $\mathcal{B}$ of **small** jobs
  - arrival rate λ
  - an average execution time ε much smaller than that of the large jobs.

Per job

The rate of backfill work for each job is given by $\zeta = \mathrm{Z} \cdot \frac{p}{P} = \lambda\varepsilon \cdot \frac{p}{P}$

Optimal Reservation Sequence

Goal: minimize each job's expected cumulative execution time

Total execution time for a job walltime of X is $T_1 = \sum_{i=1}^{m} v_{\pi(i)} + X$

- During which $T_2 = \zeta T_1$ units of backfilling work are accumulated
- The total amount of backfilling work: $\sum_{x=1}^{\infty} \zeta^x T_1 = \frac{\zeta}{1-\zeta} T_1$
- The total work: $\frac{\zeta}{1-\zeta} T_1 + T_1 = \frac{1}{1-\zeta} T_1$.

$$\boxed{T = \max\left(\sum_{i=1}^{m+1} v_{\pi(i)}, \frac{1}{1-\zeta}\Big(\sum_{i=1}^{m} v_{\pi(i)} + X\Big)\right)}$$

Optimal Reservation Sequence

Goal: minimize each job's expected cumulative execution time

Total execution time for a job walltime of X is $T_1 = \sum_{i=1}^{m} v_{\pi(i)} + X$

- During which $T_2 = \zeta T_1$ units of backfilling work are accumulated
- The total amount of backfilling work: $\sum_{x=1}^{\infty} \zeta^x T_1 = \frac{\zeta}{1-\zeta} T_1$
- The total work: $\frac{\zeta}{1-\zeta} T_1 + T_1 = \frac{1}{1-\zeta} T_1$.

$$T = \max\left(\sum_{\mathbf{i=1}}^{\mathbf{m+1}} \mathbf{v}_{\pi(\mathbf{i})}, \frac{1}{1-\zeta}\left(\sum_{i=1}^{m} v_{\pi(i)} + X \right) \right)$$

If the remaining time at the end of the last reservation exceeds all the accumulated backfilling work

Optimal Reservation Sequence

Goal: minimize each job's expected cumulative execution time

Total execution time for a job walltime of X is $T_1 = \sum_{i=1}^{m} v_{\pi(i)} + X$

- During which $T_2 = \zeta T_1$ units of backfilling work are accumulated
- The total amount of backfilling work: $\sum_{x=1}^{\infty} \zeta^x T_1 = \frac{\zeta}{1-\zeta} T_1$
- The total work: $\frac{\zeta}{1-\zeta} T_1 + T_1 = \frac{1}{1-\zeta} T_1$.

$$T = \max\left(\sum_{i=1}^{m+1} v_{\pi(i)}, \frac{1}{1-\zeta}\left(\sum_{i=1}^{m} v_{\pi(i)} + X \right) \right)$$

If the remaining time is not enough
to execute all the backfilling work

Multiple job scenario

Greedy approach choosing jobs in non-increasing order of $p_j \cdot t_{j,1}$.

- Each job j is using a reservation of $t_{j,1}$ for the first submission
- In case of failure the job is resubmitted with $t_{j,2}$ and so on

Multiple job scenario

Greedy approach choosing jobs in non-increasing order of $p_j \cdot t_{j,1}$.

- Each job j is using a reservation of $t_{j,1}$ for the first submission
- In case of failure the job is resubmitted with $t_{j,2}$ and so on

TOptimal: When $\zeta == 0$: $T_{j,i}$ are equal to Gopi's talk
ATOptimal: When $\zeta == 1$: $T_{j,1}$ is the maximum reservation

Multiple job scenario

TOptimal: When $\zeta == 0$: $T_{j,i}$ are equal to Gopi's talk
ATOptimal: When $\zeta == 1$: $T_{j,1}$ is the maximum reservation

Algorithm	Sequence of requests (in hours)
TOptimal	10.8, 13.4, 15.4, 17.1, 18.7, 20.0
ATOptimal ($\zeta = 0.1$)	10.86, 13.91, 18.69, 20.0
ATOptimal ($\zeta = 0.5$)	13.04, 20.0
ATOptimal ($\zeta = 0.9$)	17.39, 20.0
ATOptimal ($\zeta = 1$)	20.0

Truncated Normal distribution bounded by 0 and 20 hours, $\mu = 8$, $\sigma = 2$

Speculative backfilling

Backfill a job even if its reservation is larger than needed

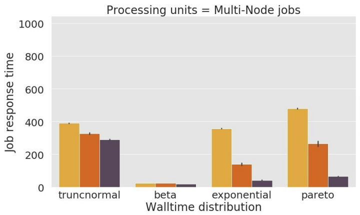
- Choose the job that maximizes the expected utilization of the gap as follows
- In case the job fails it will retake the same place in the waiting queue

For a gap of q processors and d duration:

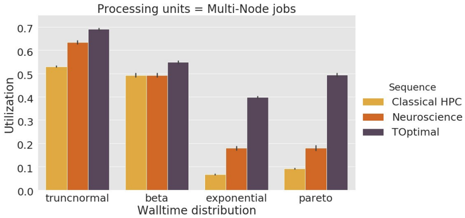
$$\max_{J_j \in \mathcal{J}'} G_j = \frac{p_j \int_{a'_j}^{d} t \cdot f'_j(t) dt}{q \cdot d}$$

a'_j and $f'_j(t) = f_j(t | t \geq a'_j)$ are the updated lower bound and PDF of the job

Simulation results



Average job response time



System utilization

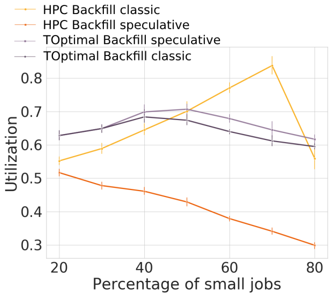
System utilization and average job response time under different walltime distributions for jobs whose processor allocations follow the Beta distribution

Neuroscience uses the last few runs to decide the requested time and 1.5x increase factor in case of failures

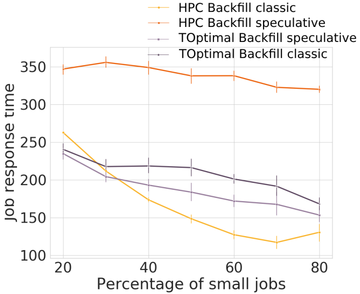
Speculative backfilling

Varying the percentage of smaller jobs within the total number of jobs

- Small improvement for TOptimal compared to HPC
- Speculative HPC exceeds TOptimal for high number of small jobs



Utilization

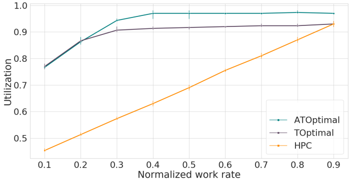


Average job response time

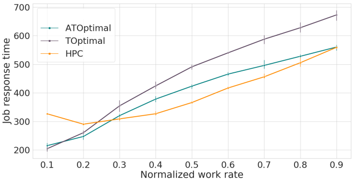
Incoming rate of small jobs

Varying ζ from 0.1 to 0.9

- Results for ATOptimal move between TOptimal ($\zeta = 0$) and HPC
- The utilization of the machine is always better using ATOptimal
- Response time is better than Toptimal but worse than HPC



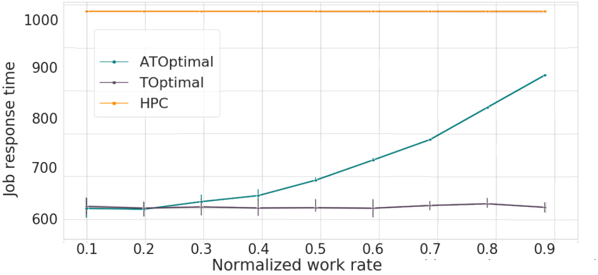
Utilization



Average job response time

Incoming rate of small jobs

Average response time only for large jobs when varying the normalized work rate for backfilling jobs ζ



Simulating neuroscience on Intrepid

- Normalized rate of backfilling work ($\zeta = 0.21$)

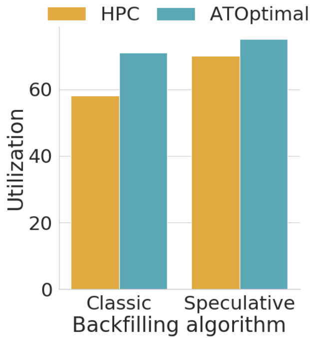
Application	Abdominal multi-organ segmentation
Distribution	Truncated Normal from 11 to 31 hours
Parameters	$\mu = 20$ and $\sigma = 8$
# Submissions	10

Application	Whole brain segmentation and cortical reconstruction
Distribution	Truncated Normal from 1.5 to 3 hours
Parameters	$\mu = 1.7$ and $\sigma = 0.5$
# Submissions	90

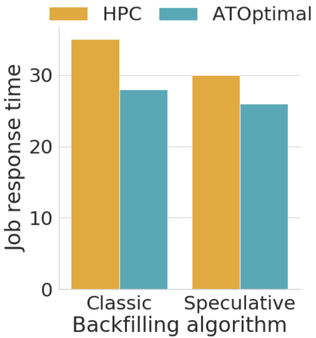
Application	FSL library of MRI and DTI analysis tools
Distribution	Truncated Normal from 10 to 35 minutes
Parameters	$\mu = 20$ and $\sigma = 8$ minutes
# Submissions	300

Simulating neuroscience on Intrepid

Simulating two weeks of neuroscience applications' execution on Intrepid



Utilization



Average job response time

Conclusion

Speculative scheduling can be integrated on top of existing HPC schedulers

- Job response time is decreased by 25%
- Processor idle time decreases, processor busy not-useful time increases
- Overall effective utilization increases by 30%

Users can opt-in and either provide past behavior to the scheduler or provide storage space for the scheduler to store execution logs

Speculative checkpointing useful only if not all applications opt-in

Future work

Multi-resource scheduling: memory, # processors

Checkpoints at the end of some/all reservations

Implementation issues

- What can users provide to schedulers?
- Impact on power consumption?
- What is the overhead?