Speculative Scheduler for converging HPC/Big Data/ML

Guillaume Pallez (Aupy),
Inria, Univ. Bordeaux

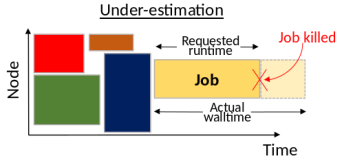
In collaboration with:

Brice Goglin, Valentin Honoré	Inria, Univ. Bordeaux
Ana Gainaru	Oak Ridge National Lab.
Padma Raghavan, Hongyang Sun	Vanderbilt University
Yves Robert	ENS Lyon, UT Knoxville

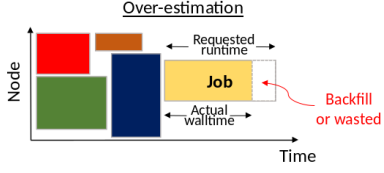
JLESC 2020

Reservation-based batch schedulers:

- Relies on (reasonably) accurate runtime estimation from the user/application
- Two queues: (i) large (main) jobs; (ii) small jobs used for backfilling.
- **Cost to users:** Pay what you use. → need to guarantee that the time asked is sufficient.



- Job killed, need to resubmit; additional cost to user.
- Waste of system resources.



- Job completed early (?).
- May waste system resources (if no backfilling possible).

Motivational examples

Sysadmin: “I want to sell all the compute slots on my platform”

User: “ I don’t want to pay if I don’t use”

Sysadmin: “Sure, then you only pay what you use...
... but, if you under-utilize your slots, you lose priority in the future.”

Motivational examples

Sysadmin: “I want to sell all the compute slots on my platform”

User: “ I don’t want to pay if I don’t use”

Sysadmin: “Sure, then you only pay what you use...
... but, if you under-utilize your slots, you lose priority in the future.”

User has one job J_1 whose execution time is **exactly 50h**.

- What does User do?
- Is Sysadmin happy?

User has one job J_2 whose execution time is **between 46h and 54h**.

- What does User do?
- Is Sysadmin happy?

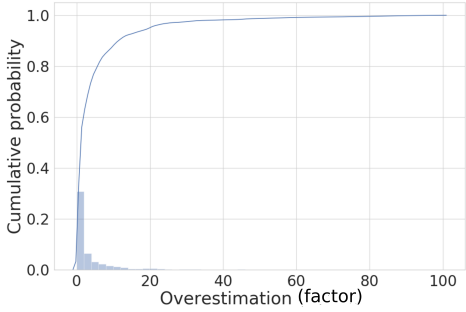
User has one job J_3 whose execution time is **between 2h and 98h**.

- What does User do?
- Is Sysadmin happy?

Anecdotal?

Study of application data from Intrepid (2009 ANL system) (data from Parallel Workload Archive).

Average job size	880 nodes / 3089 node hours
Average small jobs size	48.6 nodes / 31 node hours
Over-estimated submissions	82.2 %
Under-estimated submissions	17.7%
Average over-estimation space	2132 node hours
Percentage of small jobs	30.8%



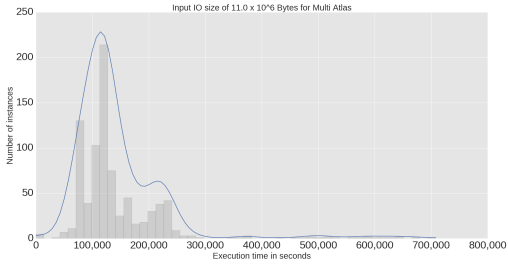
$$\text{factor} = \frac{\text{estimate - walltime}}{\text{walltime}}$$

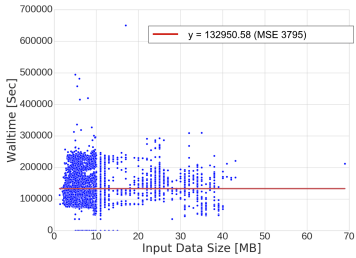
$\Longrightarrow$ **Unused backfilling space: 2.8 hours/day**

Stochastic applications

"Second generation" of HPC applications (BigData, ML) with heterogeneous, dynamic and data-intensive properties.

- Execution time is *input dependent*
- Unpredictable even for *same input size*
- Large variations

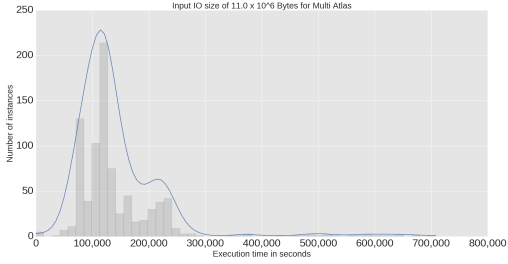


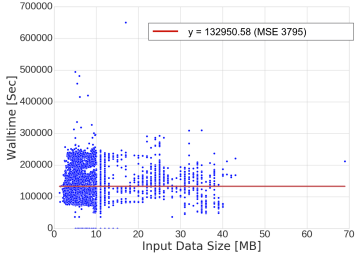


Stochastic applications

"Second generation" of HPC applications (BigData, ML) with heterogeneous, dynamic and data-intensive properties (see Session STM1.2, Honoré + Friedmann's talks).

- Execution time is *input dependent*
- Unpredictable even for *same input size*
- Large variations





Motiv: Neuroscience on HPC

Neuroscience User: “When I run on an HPC cluster, there is a very high-overhead in waiting time for each reservation. This is particularly frustrating if one of the reservation is very short.”

NU also: “When I run on the cloud, it’s much cheaper to reserve a fixed block of time, than to request a non-fixed time slot.”

Motiv: Neuroscience on HPC

Neuroscience User: "When I run on an HPC cluster, there is a very high-overhead in waiting time for each reservation. This is particularly frustrating if one of the reservation is very short."

NU also: "When I run on the cloud, it's much cheaper to reserve a fixed block of time, than to request a non-fixed time slot."

"How about checkpointing?"

- To be honest: not that easy for application users (sorry Bogdan ☺).
- Even if it was, still need to derive a reservation strategy: we don't want too many request (checkpointing+waiting time overhead)

But yes, an important tool!

Reservation-based Approach

Given a job J of duration $\mathbf{t}$ (unknown). The user makes a reservation of time t_1. Two cases:

- $\mathbf{t} \leq t_1$ The reservation is enough and the job succeeds.
- $\mathbf{t} > t_1$ The reservation is not enough. The job fails. The user needs to ask for another reservation $t_2 > t_1$.

A **strategy** is a sequence of such reservations.

For J_3 (exec **2h to 98h**):
• **Strategy:** $t_1 = 5h$, $t_2 = 40h$, $t_3 = 60h$, $t_4 = 98h$.

If the job is 33h:
1. We run the $5h$ reservation; it fails.
2. **Then** we run the $40h$; it succeeds.

Is the sysadmin happy? Is the user happy?

Reservation-based Approach

Given a job J of duration $\mathbf{t}$ (unknown). The user makes a reservation of time t_1. Two cases:

- $\mathbf{t} \leq t_1$ The reservation is enough and the job succeeds.
- $\mathbf{t} > t_1$ The reservation is not enough. The job fails. The user needs to ask for another reservation $t_2 > t_1$.

A **strategy** is a sequence of such reservations.

For J_3 (exec **2h to 98h**):
• **Strategy:** $t_1 = 5h$, $t_2 = 40h$, $t_3 = 60h$, $t_4 = 98h$.

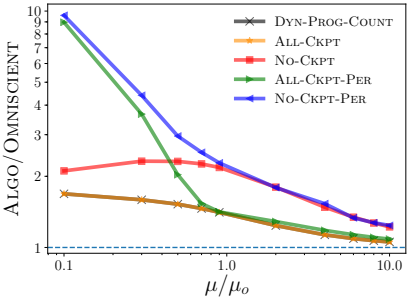
If the job is 33h:
1. We run the $5h$ reservation; it fails.
2. **Then** we run the $40h$; it succeeds.

Is the sysadmin happy?
Util: 33/45 instead of 33/98

Is the user happy?
Cost: 38 instead of 33.

Single job scheduling [Reservation and Checkpointing Strategies for Stochastic Jobs, IPDPS'20.]

We can compute in polynomial time an optimal (Dyn-Prog-Count) strategy to minimize the total expected user cost (even with checkpointing).



Appli. interpolated by LogNormal distrib ($\mu_0 = 21$h, $\sigma_0 = 19$h, $C = R = 10$min). Evolution of solutions when μ varies.

- When ratio Variance over Mean (σ/μ) increases (wide distributions), periodic strategies are arbitrarily bad.
- **BUT** Performance to omniscient seems bounded (even without checkpoint)!

Result 2: Multiple parallel jobs

Q: Now that we *know* how to deal with one job, how do we deal with multiple parallel jobs?

Result 2: Multiple parallel jobs

Q: Now that we *know* how to deal with one job, how do we deal with multiple parallel jobs?

- First phase: compute a reservation strategy for each job J_i of a batch using Result 1: $\{t_{i,1}, t_{i,2}, \dots\}$.
- Second phase: *usual* reservation scheduling:
  1. For all jobs of the batch, submit to the scheduler their smallest reservation ($\forall i,\ t_{i,1}$).
  2. Let the scheduler compute its schedule the usual way
  3. In case of $t_{i,1}$ is not enough, J_i is resubmitted with $t_{i,2}$
  4. The scheduler computes a new schedule with all resubmitted $t_{i,2}$ and so on.

Result 2: Multiple parallel jobs

Q: Now that we *know* how to deal with one job, how do we deal with multiple parallel jobs?

- First phase: compute a reservation strategy for each job J_i of a batch using Result 1: $\{t_{i,1}, t_{i,2}, \dots\}$.
- Second phase: *usual* reservation scheduling:
  1. For all jobs of the batch, submit to the scheduler their smallest reservation ($\forall i,\ t_{i,1}$).
  2. Let the scheduler compute its schedule the usual way
  3. In case of $t_{i,1}$ is not enough, J_i is resubmitted with $t_{i,2}$
  4. The scheduler computes a new schedule with all resubmitted $t_{i,2}$ and so on.

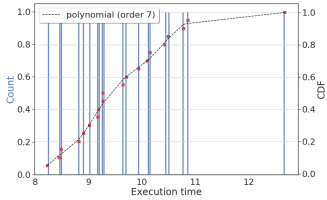
Multiple parallel job scheduling [ICPP'19]

Experimentally, this strategy seems to improve both response time and machine utilization! Can also include a backfilling model.

Result 3: robustness? (I)

Q: It seems to work, but how do you know the job distribution?

- Dealing with incomplete/low volume of information?
- Here, we consider we know k previous runs for instance.
- **Solution**: interpolation of a distribution function over those k elements, then used with algo of Result 1.



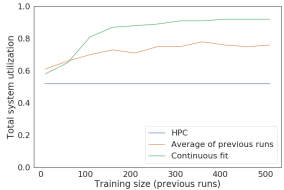
From a discrete cumulative to a continuous cumulative function, for $n = 20$ data points, sampled from a truncated normal distribution.

Result 3: robustness? (no checkpoints)

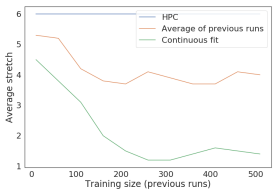
Without checkpoint

- Good performance starting with $k = 10 - 50$
- Almost best efficiency from $k \approx 150$.

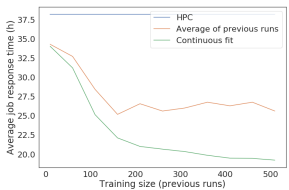
- Simulating one week in a life of an HPC scheduler (data from Intrepid for submission information, rate, occupation).
- 6 models of applications based on Neuroscience app.



Utilization
(up is good)



Stretch
(down is good)

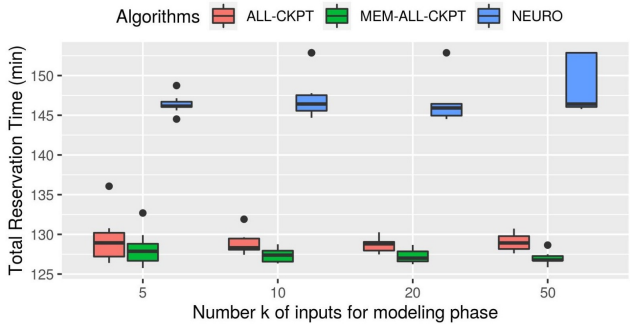


Response Time
(down is good)

Result 3: robustness? (checkpoints)

With checkpoint [under submission]

- Almost optimal performance with $k = 5$!
- Very strong robustness!



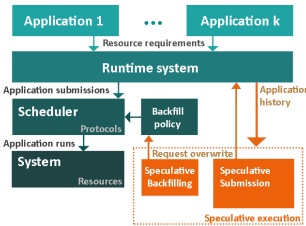
Conclusions

Pay what you use is not a viable solution for HPC system with the next generation of applications (or need lots of backfilling).

$\Longrightarrow$ Low system utilization, high response time.

We propose to introduce *Speculative Scheduling* on top of existing HPC schedulers.

- Seems to improve both Job response time and effective utilization significantly!
- Needs more experiments for evalutaion
- Processor idle time decreases, **but** wasted computations increase ☹
  (although, a lot less with checkpointing)



Discussion (I)

Maybe it is time to start thinking about our models differently.

From our application models..

- Do we still want to work on perfect prediction models that needs 100k parameters, 5k h of precomputation to get something somewhat precise?
- Can we ever get a precise runtime estimate? Notion of *Input Variance*?

Or do we change entirely the way we do scheduling and accept this variability in performance?

Discussion (II)

... To our performance models

From this type of plot:

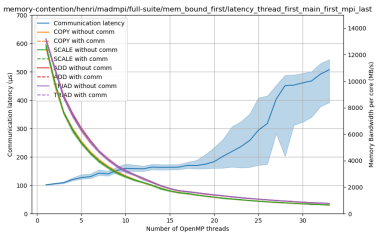


Figure Credit:

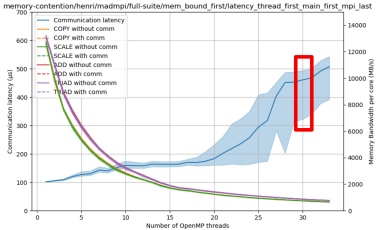
P. Swartvagher, Memory bandwidth interference models (work in progress)

Thanks

Discussion (II)

... To our performance models

From this type of plot:



To this:

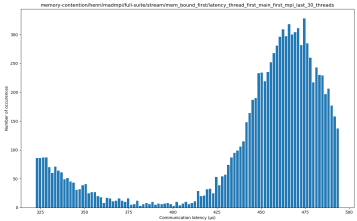


Figure Credit:

P. Swartvagher, Memory bandwidth interference models (work in progress)

Thanks