Scheduling Algorithms for Variable Capacity Resources

Anne Benoit, Lucas Perotin, Yves Robert
LIP, Ecole Normale Supérieure de Lyon, France

Rajini Wijayawardana, Chaojie Zhang, Andrew Chien
University of Chicago, USA

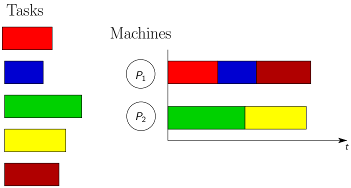
http://graal.ens-lyon.fr/~abenoit/

Scheduling Variable Capacity Resources for Sustainability
Paris Center of the University of Chicago
March 29, 2023

Motivation

- Online scheduling techniques: at the heart of *batch schedulers*
- Schedule independent jobs on parallel HPC platforms

- **Optimization objectives:**
  - Utilization (platform owner's perspective)– fraction of time where platform resources execute computations
  - Stretch (user's perspective) – minimize the maximum (or sometimes average) stretch of jobs, defined as the response time normalized by the job length



Motivation: Variable capacity

- Green computing: total available power evolves with time (cost, wind or solar energy, ...)
- How to efficiently schedule when variations in power supply imply changes in the number of available computing resources over time?
- Need to be prepared to variations: if a machine is shut down, all its jobs must be re-executed

- Design of risk-aware strategies that assign incoming jobs to the *right* target machine, for our optimization criteria
- Platform utilization no longer an adequate criterion (partial executions of jobs that get killed do not count as actual progress of the jobs) $\Rightarrow$ Goodput – *useful* platform utilization, accounting only for jobs that are running or have completed

Outline

1. Framework and complexity
2. Heuristics
3. Simulations
4. Conclusion

Platform and jobs

Platform:

- Set $\mathcal{M}$ of M^+ identical parallel machines, each equipped with n_c cores, and requiring power P when switched on
- Overall available power capacity $P(t)$: function of time t (time discretized) $\Rightarrow$ $M_{alive}(t)$ machines alive
- $b_{m,t}$: boolean decision variable, equal to 1 if machine m is active at time t and 0 otherwise: $\forall t, \sum_{m \in \mathcal{M}} b_{m,t} \times P \leq P(t)$

Jobs:

- Set $\mathcal{J}$; job $\tau_i \in \mathcal{J}$ released at date r_i, needs c_i cores, has length w_i; allocated to machine m_i at starting date s_i
- (Predicted) completion date of job τ_i: $e_i = s_i + w_i$ if not interrupted
- At any time, cores used by running jobs on a machine $\leq n_c$

Objective function: Goodput

- $\mathcal{J}_{comp,T}$: set of jobs that are complete at time T ($e_i \leq T$)
- $\tau_i \in \mathcal{J}_{started,T}$: set of jobs running and not dead at time T ($s_i \leq T < e_i$)
- Total number of units of work that can be executed in $[0, T]$: at most $\sum_{t \in [0,T-1]} M_{alive}(t) n_c$,

- $\textsc{Goodput}(T)$ – fraction of useful work up to time T:

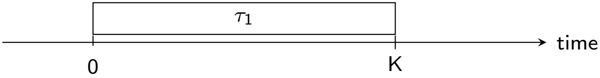
$$\textsc{Goodput}(T) = \frac{\sum_{\tau_i \in \mathcal{J}_{comp,T}} w_i c_i + \sum_{\tau_i \in \mathcal{J}_{started,T}} (T - s_i) c_i}{n_c \sum_{t \in [0,T-1]} M_{alive}(t)}$$

Complexity result

Theorem

An adversary can force any schedule to achieve no goodput at all, even with a single unicore machine.

- Job τ_1 of size $c_1 = 1$ and duration $w_1 = K$ released at time $t = r_1 = 0$; Goodput of the machine at time $T = K$



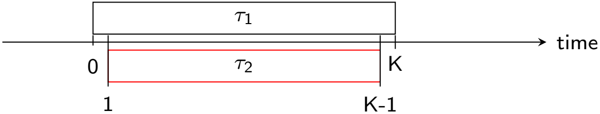
- Start τ_1 at time $s_1 > 0$: machine interrupted at time K

Complexity result

Theorem

An adversary can force any schedule to achieve no goodput at all, even with a single unicore machine.

- Job τ_1 of size $c_1 = 1$ and duration $w_1 = K$ released at time $t = r_1 = 0$; Goodput of the machine at time $T = K$



- Start τ_1 at time $s_1 = 0$: new job τ_2, machine interrupted at time $K - 1$

Outline

1. Framework and complexity
2. Heuristics
3. Simulations
4. Conclusion

Events

Algorithms: Take action whenever an *event* occurs

- **Job Arrival Event** – Job released: decide when to schedule it and on which machine
- **Job Completion Event** – Job completed: release the cores it was using, possibly allowing for additional jobs to be scheduled
- **Machine Addition Event** – New machine available: decide how to utilize it
- **Machine Removal Event** – Machine switched off: kill jobs and decide how to reallocate them

Different heuristics take different decisions

FirstFitAware and FirstFitUnaware

Baseline heuristics:

- Machines labeled from 1 to M^+; jobs scheduled on the machine with the smallest available index that has enough free resources to execute it
- Use of waiting queue for pending jobs
- When a machine needs to be switched off, FirstFitAware kills the machine with the highest index

- FirstFitUnaware: Not aware of the risk of shutdown incurred by the machines, and hence switches off randomly a machine rather than ordering them by index

TargetStretch

- Interrupting long job ⇒ significant work loss
- Schedule smaller jobs on machines that are likely to be turned off (large indices), and longer jobs on machines that will never be turned off (small indices)
- Consider a target stretch value, and one queue per machine
- For the TargetStretch heuristic, at each **Job arrival event**: compute the job's target machine; consider neighboring machine if target stretch not achievable
- Set of risk-free machines recomputed at machine addition/removal events, and jobs might be reallocated

TargetASAP and PackedTargetASAP

- In TargetStretch, with large target stretch, bad utilization as job goes to target machine; no flexibility to go to another free machine
- TargetASAP proposes a new strategy at **job arrival event**:
  - try to start job immediately on target machine or on closest machine in the neighborhood;
  - if not possible, assign on target machine if target stretch not exceeded;
  - otherwise, assign on machine where it can start ASAP (within acceptable distance)
- Variant PackedTargetASAP: group machines per packs, and assign jobs to first machines of the pack, to leave machines empty for future jobs with large number of cores

Outline

1 Framework and complexity

2 Heuristics

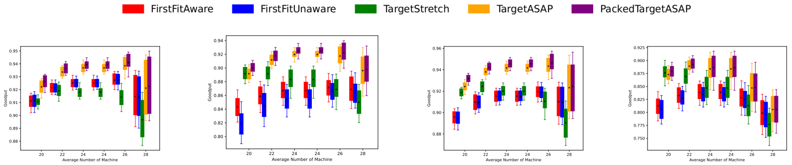
3 Simulations

4 Conclusion

Simulation setting

- In-house simulator, using a combination of two traces:
  - Resource variation trace representing the number of machines alive at any given time – Use of a random walk, within an interval
  - Job trace:
    - Real traces coming from **Borg** (two-week traces with jobs coming from **Google** cluster management software: release dates, lengths, number of cores)
    - Synthetic traces to study the impact of parameters (three variants: uniform lengths, log scale, and three types of jobs)

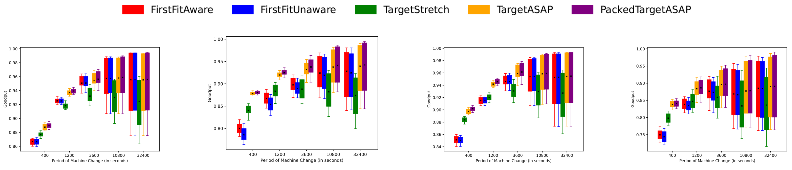
Varying the number of machines



(a) SyntheticUniform (b) SyntheticLogScale (c) Synthetic3types (d) Borg

- FirstFitAware and FirstFitUnaware never good
- TargetStretch: different behavior because of its lack of flexibility, some machines remain partially inactive even when jobs are waiting (better with fewer machines)
- TargetASAP always good, and the packed variant PackedTargetASAP even better

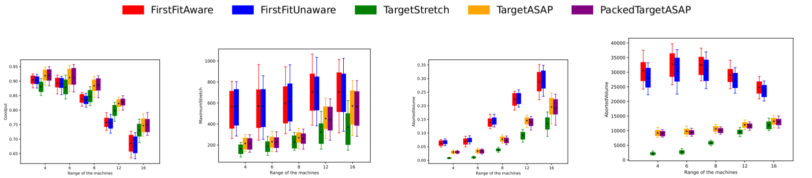
Varying the period of machine variation



(a) SYNTHETICUNIFORM (b) SYNTHETICLOGSCALE (c) SYNTHETIC3TYPES (d) BORG

- As before, limited impact of workflow
- With low period (many changes), TARGETSTRETCH better by preserving long jobs
- Goodput increases with the period: less changes means less job interruptions
- More impact of new TARGETASAP and PACKEDTARGETASAP strategies with high variability (low periods)

Exploring other metrics



(a) GOODPUT (b) MAXIMUMSTRETCH (c) ABORTEDVOLUME (d) AVERAGEABORTEDTIME

Different metrics to analyze the results for BORG (varying the range of the machines)

- Increase in range $\Rightarrow$ Degradation of the metric
- TARGETSTRETCH achieves the lowest maximum stretch, as well as low aborted volume and time
- However, low utilization of machines for TARGETSTRETCH, with low goodput

Outline

1. Framework and complexity
2. Heuristics
3. Simulations
4. Conclusion

Conclusion

- Right in scope of the workshop: Scheduling with variable capacity resources
- Formalization of the problem, model and objective functions
- First attempt at providing practical solutions to the problem
- TargetStretch very good to minimize maximum stretch, but leads to a poor resource utilisation
- Clever strategies TargetASAP and PackedTargetASAP achieve very good goodput

- On-going collaboration, looking forward to new ideas emerging from discussions these days ☺