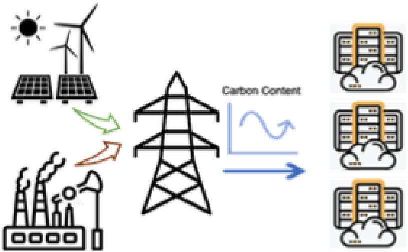
Scheduling Algorithms for Variable Capacity Resources

Anne Benoit
LIP, Ecole Normale Supérieure de Lyon
Institut Universitaire de France

Joint work with Y. Robert, L. Perotin, J. Cendrier, F. Vivien (ENS Lyon)
and A. A. Chien, R. Wijayawardana, C. Zhang (U. Chicago)

May 16, 2024 – New Challenges in Scheduling Theory – Aussois

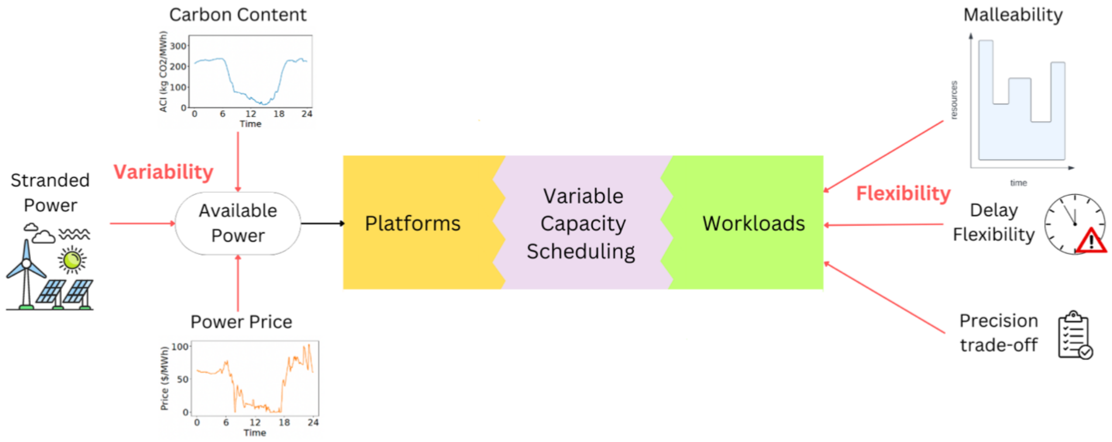
Variable power



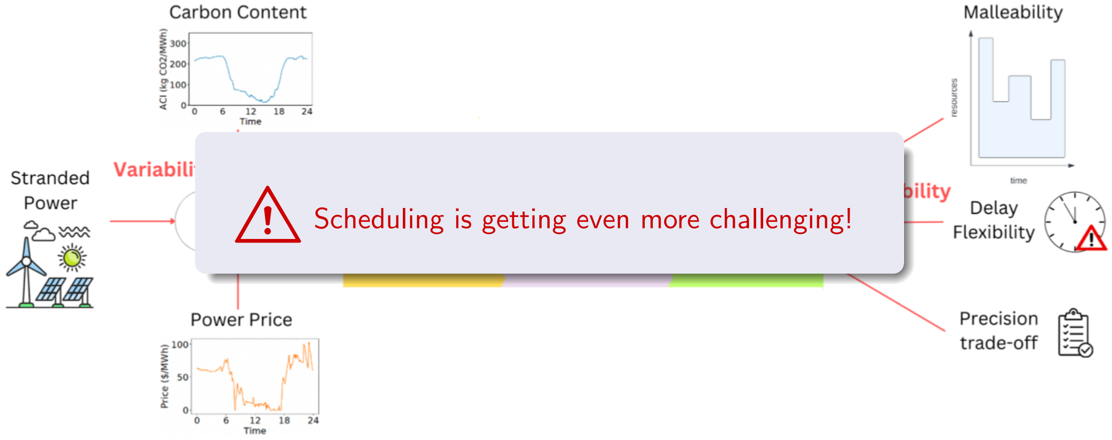
- Today's data centers assume resource capacity as a fixed quantity
- Emerging approaches:
  - Exploit grid renewable energy
  - Reduce carbon emissions

⇒ Variable power

Big picture


Carbon Content
ACI (kg CO2/MWh)
300
200
100
0
0 6 12 18 24
Time
Stranded Power
Variability
Available Power
Power Price
Price ($/MWh)
100
50
0
0 6 12 18 24
Time
Platforms
Variable Capacity Scheduling
Workloads
Malleability
resources
time
Flexibility
Delay Flexibility
Precision trade-off

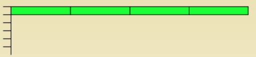
Big picture



Scheduling is getting even more challenging!

Outline

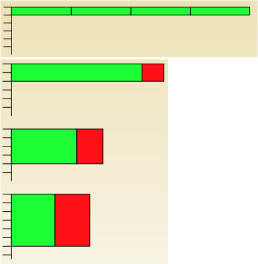
Parallel jobs



- **Rigid jobs**: Processor allocation is fixed
- **Moldable jobs**: Processor allocation is decided by the user or the system but cannot be changed during execution
- **Malleable jobs**: Processor allocation can be dynamically changed

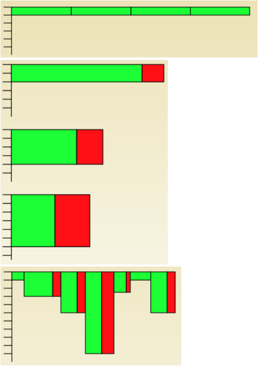
Parallel jobs

- **Rigid jobs**: Processor allocation is fixed
- **Moldable jobs**: Processor allocation is decided by the user or the system but cannot be changed during execution
- **Malleable jobs**: Processor allocation can be dynamically changed



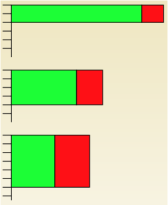
Parallel jobs

- **Rigid jobs**: Processor allocation is fixed
- **Moldable jobs**: Processor allocation is decided by the user or the system but cannot be changed during execution
- **Malleable jobs**: Processor allocation can be dynamically changed



Parallel jobs

- **Rigid jobs**: Processor allocation is fixed



- **Moldable jobs**: Processor allocation is decided by the user or the system but cannot be changed during execution
- **Malleable jobs**: Processor allocation can be dynamically changed

The case for moldable jobs:

- Easily adapt to the amount of available resources (contrarily to rigid jobs)
- Easy to design/implement (contrarily to malleable jobs)
- Computational kernels in scientific libraries are provided as moldable jobs

Checkpoints

- Some jobs cannot be interrupted
- Some jobs can be checkpointed

Half the projected load for US Exascale systems include checkpointing capabilities (from APEX worklows, Sandia/LosAlamos/NERSC report, April 2016)

Checkpoints

Scheduling opportunity

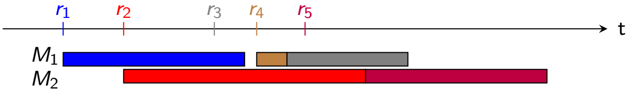
- Many checkpointable jobs are moldable
- These jobs are able to restart with a different allocation (size and shape)

Resizing impacts performance

(from APEX workflows, Sandia/LosAlamos/NERSC report, April 2016)

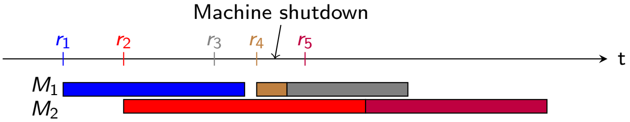
Risk aware?

1. **Which machine to shutdown?**



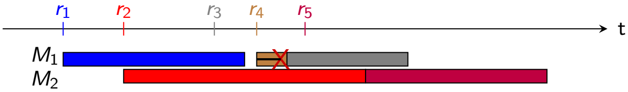
Risk aware?

1. **Which machine to shutdown?**



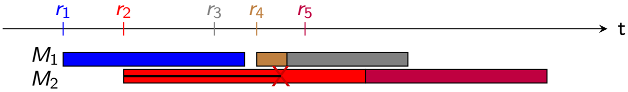
Risk aware?

1. **Which machine to shutdown?**



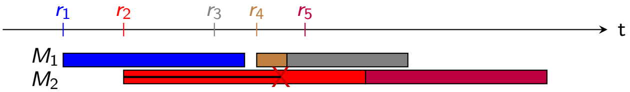
Risk aware?

1. **Which machine to shutdown?**



Risk aware?

1. **Which machine to shutdown?**



2. **How to schedule jobs to minimize impact?**

Main questions

- When power decreases, which machines to power off? Which jobs to interrupt? And to re-schedule?
- Are we notified ahead of a power change?
  - Resource variation in power obeys specific parameters whose evolution is dictated by a mix of technical availability and economic conditions
  - Accurate external predictor (precision, recall)? Maybe too optimistic ☹
- Re-scheduling interrupted jobs
  - Can we take a proactive checkpoint before the interruption?
  - Which priority should be given to each interrupted job?
  - Which geometry and which nodes for re-execution?

Main questions

- When power decreases, which machines to power off? Which jobs to interrupt? And to re-schedule?
- Are we notified ahead of a power change?

Scheduling opportunity & challenge

- Nodes ordered according to non-decreasing risk, say from left to right
- Shutdown nodes starting from the right
- Assign priority jobs, such as large jobs, to nodes on the left
- Global load of the platform must remain balanced

⚠ Sophisticated algorithms that go well beyond first-fit decisions

Outline

Platform

- Set $\mathcal{M}$ of M^+ identical parallel machines, each equipped with n_c cores, and requiring power P when switched on
- Global available power capacity $P(t)$: function of time t (time discretized) $\Rightarrow M_{alive}(t)$ machines alive, with $M_{alive}(t)P \leq P(t)$

Rigid 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, total cores used by running jobs on a machine $\leq n_c$

Resource variation

- The number of alive machines evolves over time (either random-length phases, or fixed-length periods)
- The number of alive machines in the next phase/period is not known in advance
- Technically, $M_{alive}(t)$:
  - Always ranges in interval $[M^- = M_{avg} - M_{ra}, M^+ = M_{avg} + M_{ra}]$ centered in M_{avg}
  - Evolves according to some random walk, starting with M_{avg}
  - Stays constant, increases or decreases with same probability (if range bound reached, stays constant or evolves in unique possible direction, with same probability)
  - Magnitude of variation controlled by another variable

Limitations

- Rigid jobs ⇒ no flexibility in size
- Identical multicore machines
- No checkpoints
- Power consumption at time t proportional to $M_{alive}(t)$ (actual load not accounted for)
- Resource variation not known until change

Objective function: Goodput

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

$$n_c \sum_{t \in [0, T-1]} M_{alive}(t)$$

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

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

Keep an eye on maximum stretch

Objective function: Goodput

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

$$n_c \sum_{t \in [0, T-1]} M_{alive}(t)$$

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

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

Keep an eye on maximum stretch

Complexity

Theorem

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

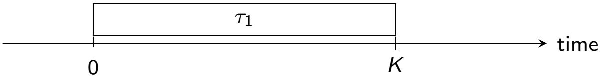
- Job τ_1 of size $c_1 = 1$ and duration $w_1 = K$ released at time $t = r_1 = 0$; Goodput of the machine at time $T = K$?

Complexity

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 $T = K$?



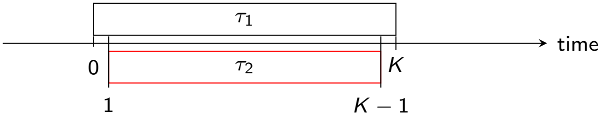
- Start τ_1 at time $s_1 > 0$: machine interrupted at time K

Complexity

Theorem

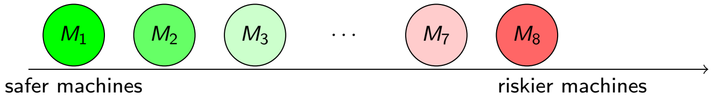
An adversary can force any schedule to achieve no goodput at all, even with a single unicore machine

- 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 $T = K$?



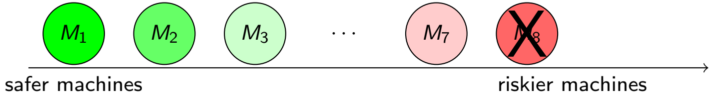
- Start τ_1 at time $s_1 = 0$: new job τ_2, machine interrupted at time $K - 1$

Risk-aware



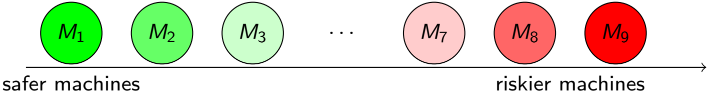
Risk-aware job allocation strategies

Risk-aware



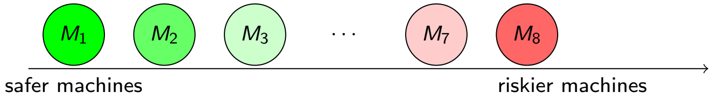
Risk-aware job allocation strategies

Risk-aware



Risk-aware job allocation strategies

Risk-aware



Risk-aware job allocation strategies

Events:

- **Job arrival**: When a job is released, when to schedule it and on which machine?
- **Job completion**: When a job is completed, its cores are released $\Rightarrow$ additional jobs can be scheduled
- **Machine addition**: When a new machine becomes available, how to utilize it?
- **Machine removal**: When a machine is turned off, its jobs are killed and need re-allocation

FirstFitAware

- **Job arrival**
  Assign incoming job to smallest-index machine with enough free resources
  If no machine can execute the job, it is placed in waiting queue
- **Job completion**
  Check the queue for job with smallest release date that fits in the machine *m* with completed job, and assigns it to *m*
  If a job is assigned, continues to search the queue
  If empty queue or not enough cores in *m* for any waiting job $\Rightarrow$ no action
- **Machine addition**
  Assign jobs to the new machine in order of increasing release date
- **Machine removal**
  Shut down machine with highest index, put all its jobs in the queue
  Assign jobs to available machines in order of increasing release date

FirstFitAware

- **Job arrival**
  Assign incoming job to smallest-index machine with enough free resources
  If no machine can execute the job, it is placed in waiting queue

Risk-aware

- Ordered list of machines
- Jobs mapped to leftmost (safer) machines whenever possible
- Rightmost (riskier) machines are shutdown whenever necessary

- **Machine addition**
  Assign jobs to the new machine in order of increasing release date
- **Machine removal**
  Shut down machine with highest index, put all its jobs in the queue
  Assign jobs to available machines in order of increasing release date

FirstFitAware

- **Job arrival**
  Assign incoming job to smallest-index machine with enough free resources
  If no machine can execute the job, it is placed in waiting queue

Risk-aware
- Ordered list of machines
- Jobs mapped to leftmost (safer) machines whenever possible
- Rightmost (riskier) machines are shutdown whenever necessary

FirstFitUnaware: Shutdown random machines whenever necessary

vith

- **Machine removal**
  Shut down machine with highest index, put all its jobs in the queue
  Assign jobs to available machines in order of increasing release date

FIRSTFITAWARE

- **Job arrival**
  Assign incoming job to smallest-index machine with enough free resources
  If no machine can execute the job, it is placed in waiting queue

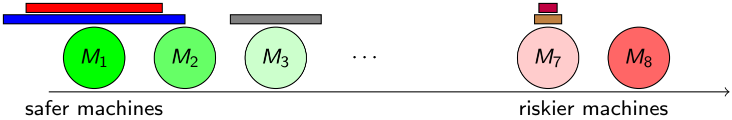
vith

Risk-aware
- Ordered list of machines
- Jobs mapped to leftmost (safer) machines whenever possible
- Rightmost (riskier) machines are shutdown whenever necessary

FIRSTFITUNAWARE: Shutdown random machines whenever necessary

Interrupting a long job is a big performance loss
Schedule smaller jobs on machines that are likely to be turned off
Schedule longer jobs on risk-free machines

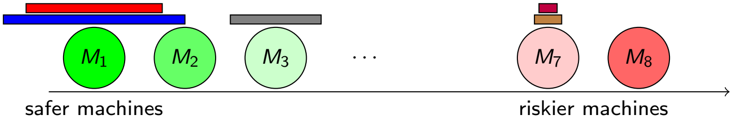
TargetStretch



- Add one queue per machine
- Set target value for (target) maximum stretch
- **Job arrival**
  Compute job's target machine
  Consider neighboring machines if target stretch not achievable
- **Machine addition/removal**
  Set of risk-free machines recomputed
  Re-allocate pending jobs

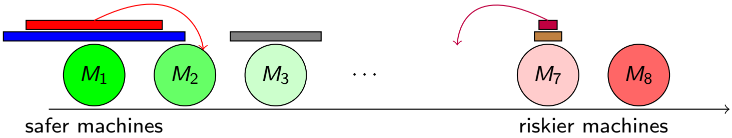
TargetASAP & PackedTargetASAP

- TargetStretch: potential bad utilization
  No flexibility for mapping to another free machine



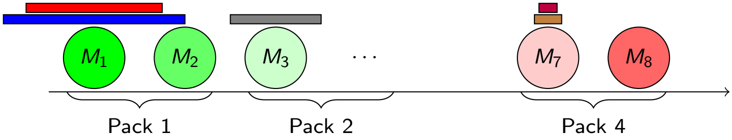
TargetASAP & PackedTargetASAP

- TargetStretch: potential bad utilization
  No flexibility for mapping to another free machine
- TargetASAP:
  - Start job immediately on target machine or closest machine in 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)



TargetASAP & PackedTargetASAP

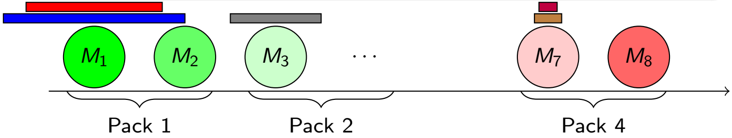
- TargetStretch: potential bad utilization
  No flexibility for mapping to another free machine
- TargetASAP:
  - Start job immediately on target machine or closest machine in neighborhood
  - If not possible, assign on target machine if target stretch not exceeded
  - Otherwise, assign on machine where it can start ASAP (within acceptable distance)
- Variant PackedTargetASAP: group machines into packs, and assign jobs to first machines of the pack, to leave machines empty for future large jobs



TargetASAP & PackedTargetASAP

- TargetStretch: potential bad utilization
  No flexibility for mapping to another free machine
- TargetASAP:
  - Start job immediately on target machine or closest machine in 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 P … ssign jobs to
  first mach … jobs

Technical and kind of painful
despite all simplifying hypotheses ☹



Simulation setting

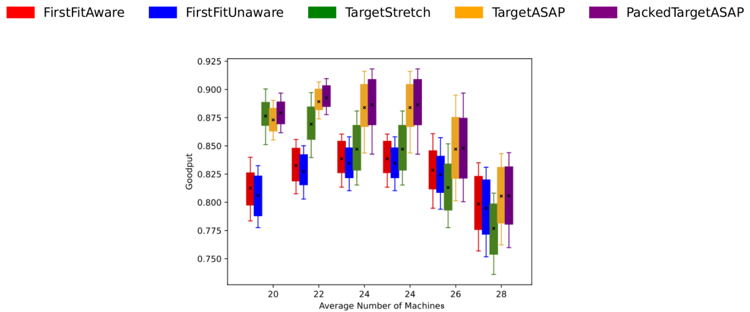
In-house simulator, using a combination of two traces:

- Resource variation trace: 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) $\Rightarrow$ similar conclusions

Dimensioning

- Number of available machines always in $[M_{avg} - M_{ra}, M_{avg} + M_{ra}]$
- Total work hours $\approx$ maximum capacity of 26 machines each with 24 cores, running during 2 weeks with full peak load
- Average number of machines: $M_{avg} = 24$
- Period of machine variation: $\phi = 20mn$
- Range of machine variation: $M_{ra} = 8$; half the machines are safe
- Number of cores per machine: $n_c = 24$. Jobs typically use $1, 2, 4, 8$ cores
- Conservative backfilling at machine level

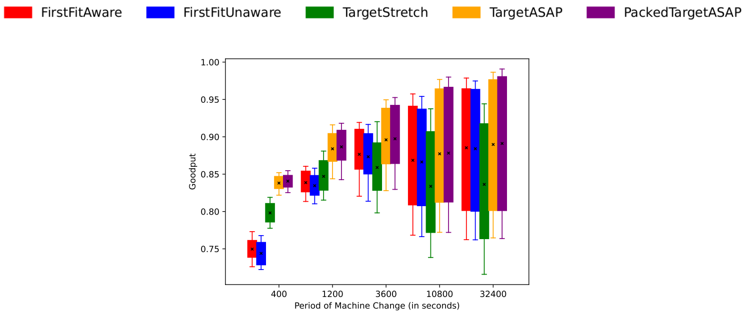
Varying the number of machines



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 packed variant PACKEDTARGETASAP even better

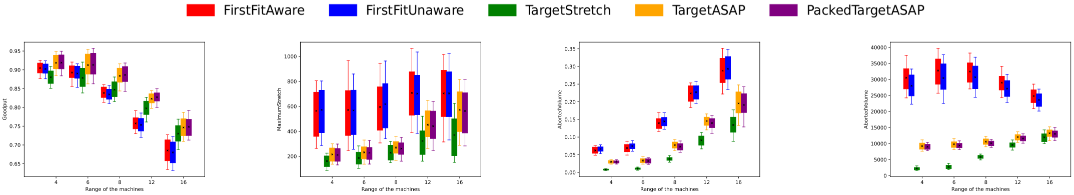
Varying the period of machine variation



Borg

- With low period (many changes), TargetStretch better by preserving long jobs
- Goodput increases with period: less changes ⇒ less job interruptions
- Better relative performance of TargetASAP and PackedTargetASAP with low periods (= high variability)

Exploring other metrics (Borg)



GOODPUT MAXIMUMSTRETCH ABORTEDVOLUME AVERAGEABORTEDTIME

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

Conclusion for this case study

- A simple case-study of scheduling with variable capacity resources
- Primary challenge: when capacity decreases, running jobs need to be terminated to meet required power load reduction
- Online risk-aware scheduling strategies to preserve performance:

  map the right job to the right machine
- Algorithmic techniques: risk index per machine, mapping longer jobs to safer machines, maintaining local queues at machines, re-executing interrupted jobs on new machines, and redistributing pending jobs as resource capacity increases
- Significant gains over first-fit algorithms with up to 10% increase in goodput, and better performance in complementary metrics (maximum and average stretch)

Outline

Model

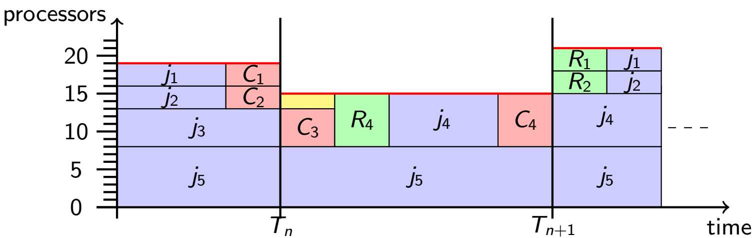
Problem: Scheduling infinite parallel rigid jobs under variable number of processors, during each *section*

Hypotheses:

- A job can be checkpointed and recovered
- Knowledge of the duration of each section, and bound on #proc difference

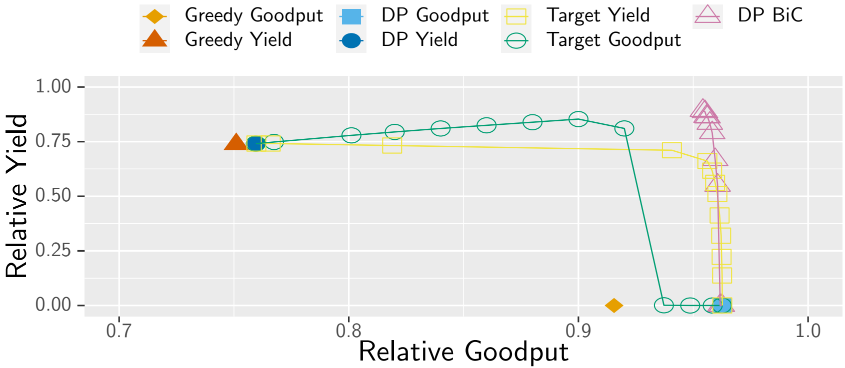
Additional constraint:

- Never lose work (i.e., checkpoint enough before section change, and never shut off a non-checkpointed job)



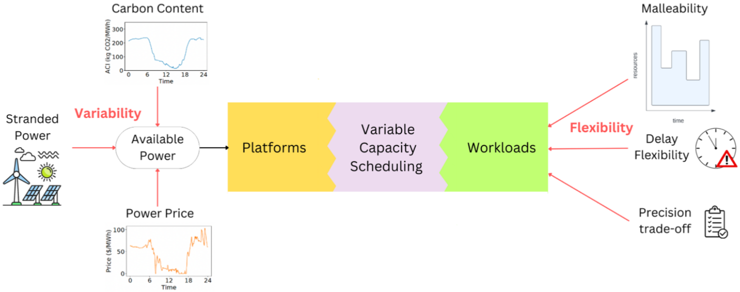
Algorithms

- Sophisticated dynamic programming algorithms to optimize goodput and/or yield at the end of a section
- Evaluation on job traces
- Improvement of novel strategies over greedy approaches



Outline

Back to the big picture



Many challenging scheduling problems ☺

Workshop report: *Scheduling Variable Capacity Resources for Sustainability*; March 29-31, 2023, U. Chicago Paris Center

Today's case study: restricted instance ☹

Risk-Aware Scheduling Algorithms for Variable Capacity Resources; PMBS workshop at SC'23

Research directions

Platforms and resources: New and more complex definitions of capacity; describe resource capacity as a function of time

Flexible workloads: Flexible start dates, allow migration or deferral

Scheduling models and metrics: New models for resource variability and job classification; New multi-criteria metrics for both performance and sustainability; Accounting for uncertainty

Policy and societal factors: Mechanisms that help people accept constraints linked to environmental rules; *Superficial feeling of abundance:* abuse of computational resources, rebound effect