The challenges of variable capacity scheduling

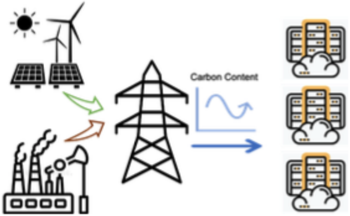
Anne Benoit
LIP, Ecole Normale Supérieure de Lyon
Institut Universitaire de France
Visiting research scholar at IDEaS

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

April 23, 2025 – IDEaS Seminar – Georgia Tech, Atlanta

Variable power

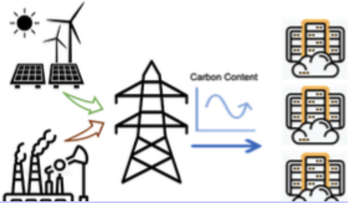
- Today's data centers assume resource capacity as a fixed quantity



- Emerging approaches:
  - Exploit grid renewable energy
  - Reduce carbon emissions

Variable power

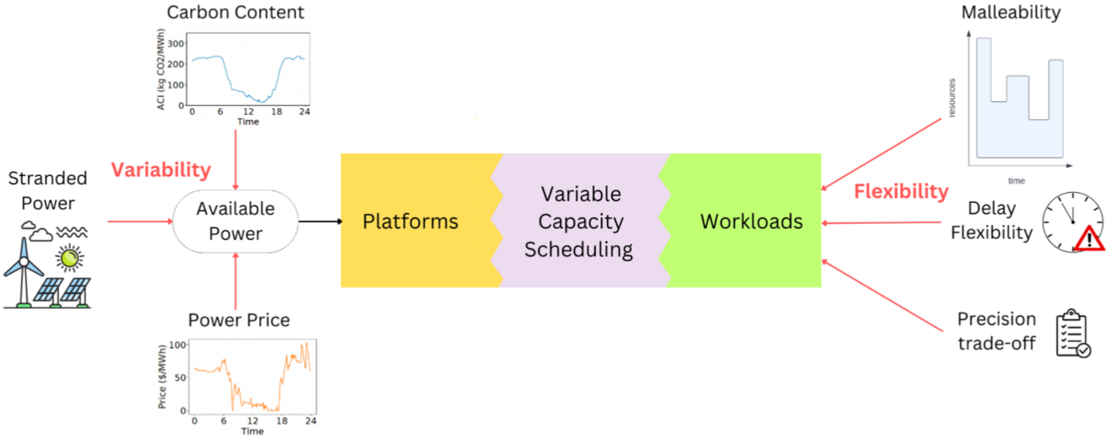
- Today's data centers assume resource capacity as a fixed quantity



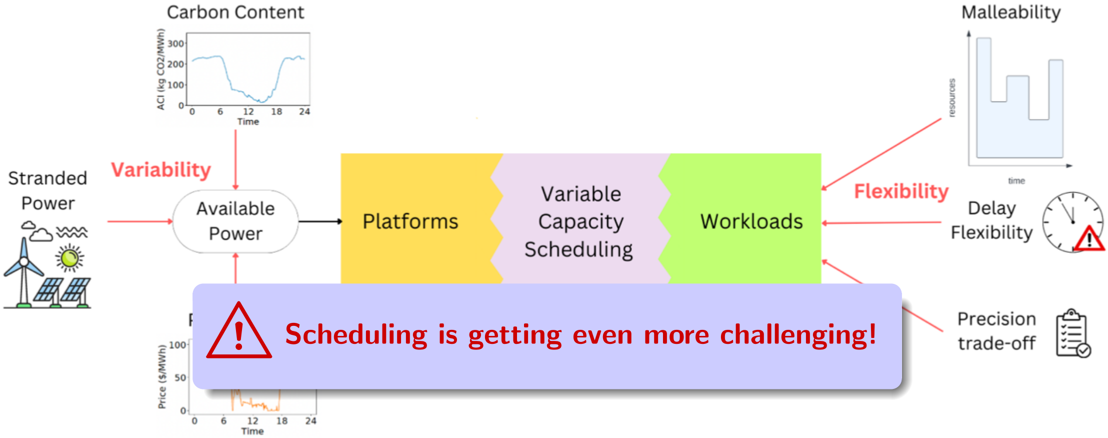
⇒ Variable power!

- Emerging approaches:
  - Exploit grid renewable energy
  - Reduce carbon emissions

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

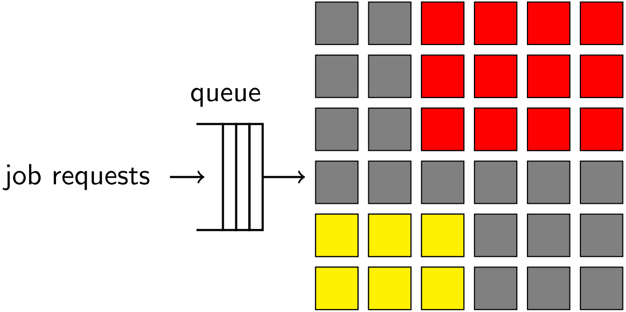


Scheduling is getting even more challenging!

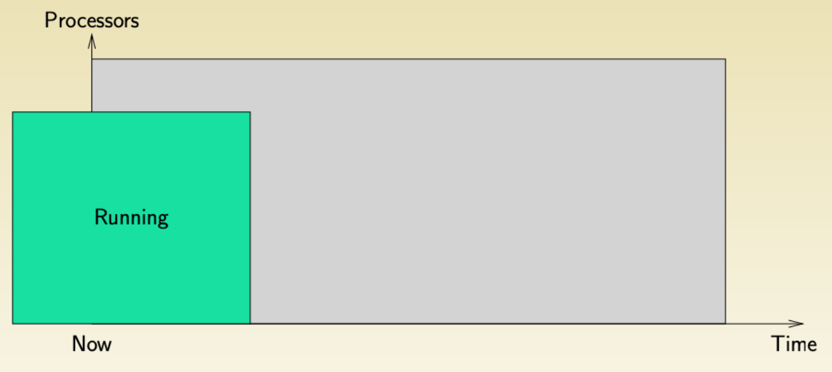
Outline

Batch scheduling

- Jobs submitted online
- Each job has a release time and a size (number of resources)
- Each job has an (estimated) execution time, a.k.a reservation length
- The *batch scheduler* is responsible for the sharing

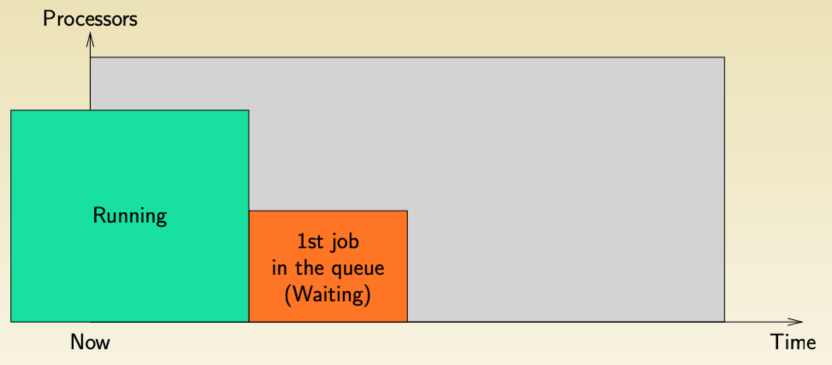


General principle



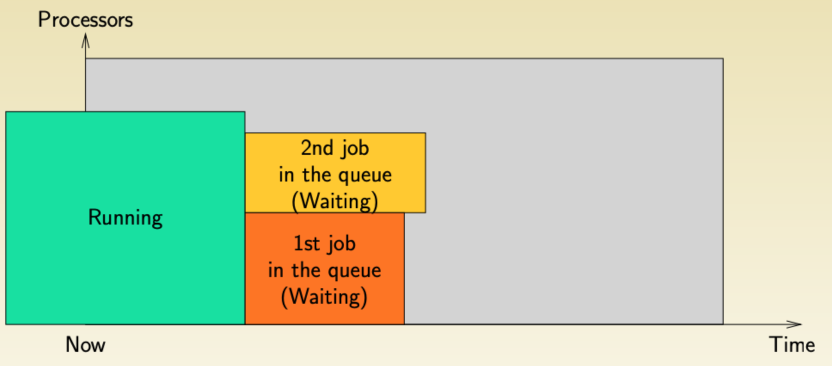
Jobs are released one after the other and are scheduled upon arrival

General principle



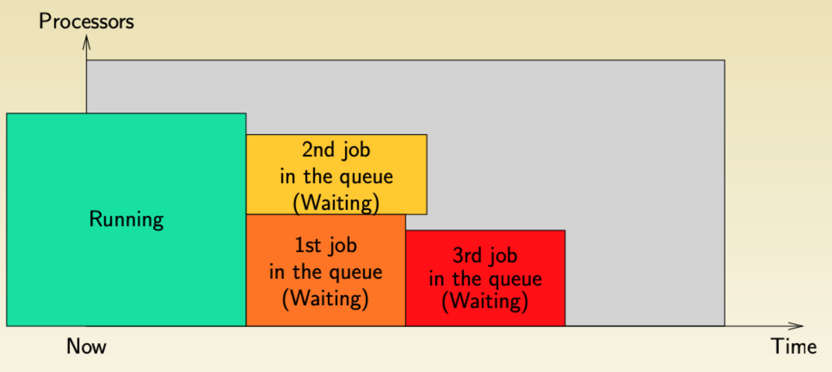
Jobs are released one after the other and are scheduled upon arrival

General principle



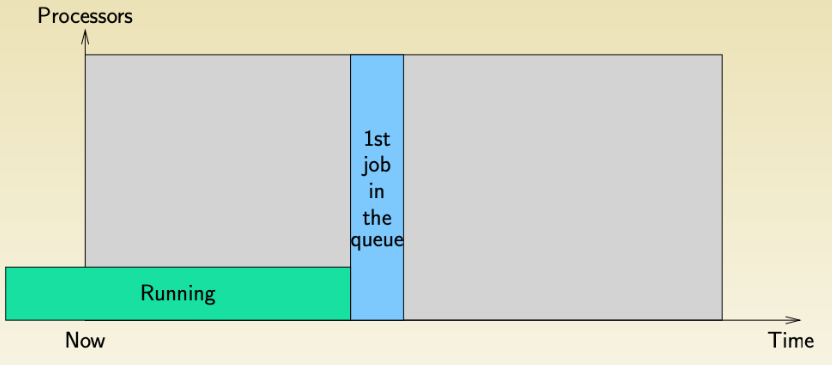
Jobs are released one after the other and are scheduled upon arrival

General principle



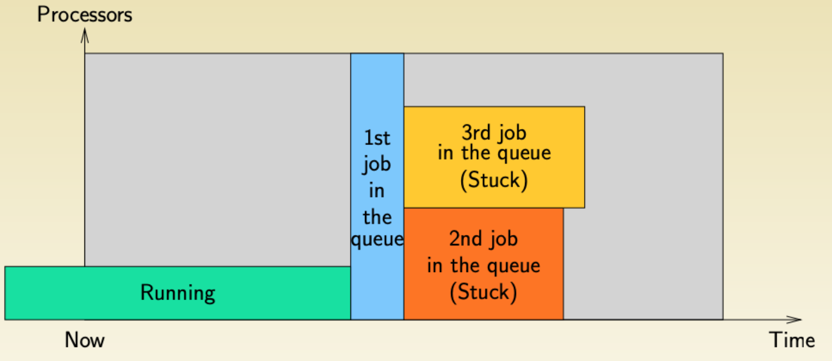
Jobs are released one after the other and are scheduled upon arrival

Backfilling



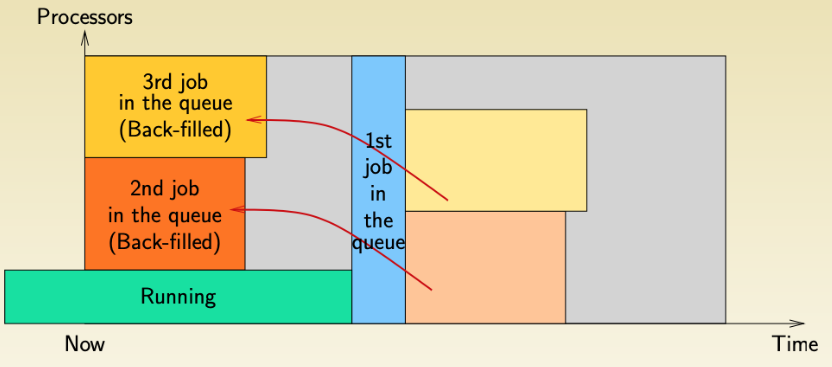
FCFS + FirstFit = simplest scheduling strategy

Backfilling



FCFS + FirstFit = simplest scheduling strategy
Fragmentation ☹

Backfilling



FCFS + FirstFit = simplest scheduling strategy

Fragmentation ☹ ⇒ need for backfilling

EASY Backfilling

Extensible Argonne Scheduling System

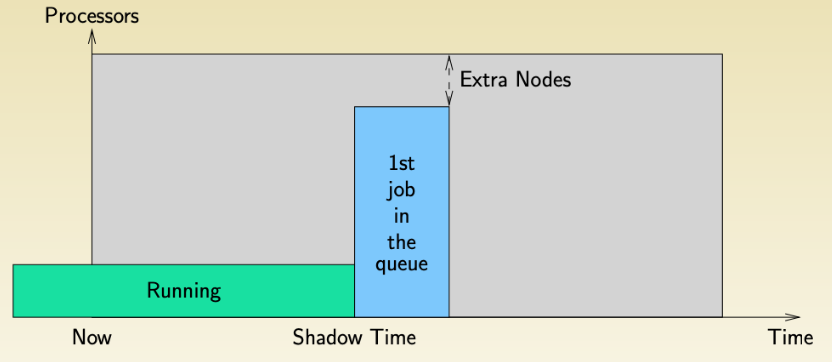
Maintain only one *reservation time*, for first job in the queue

Shadow time – Starting execution of first job in the queue

Extra nodes – Number of nodes idle at shadow time

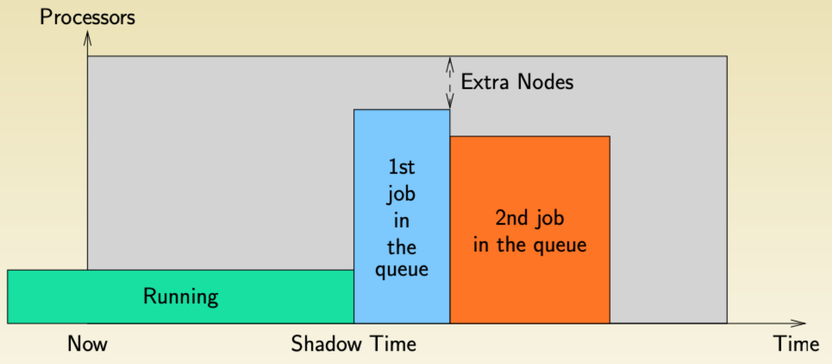
1. Go through the queue in order, starting with second job
2. Backfill a job
   - either if it will terminate by shadow time
   - or if it needs no more nodes than the extra nodes

EASY



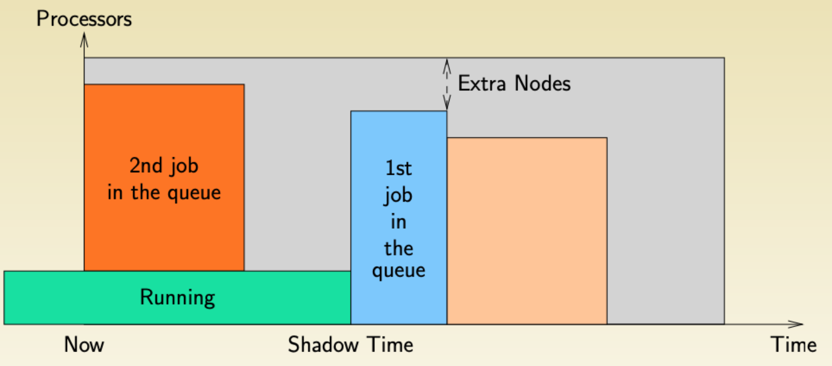
First job in the queue is never delayed by backfilled jobs

EASY



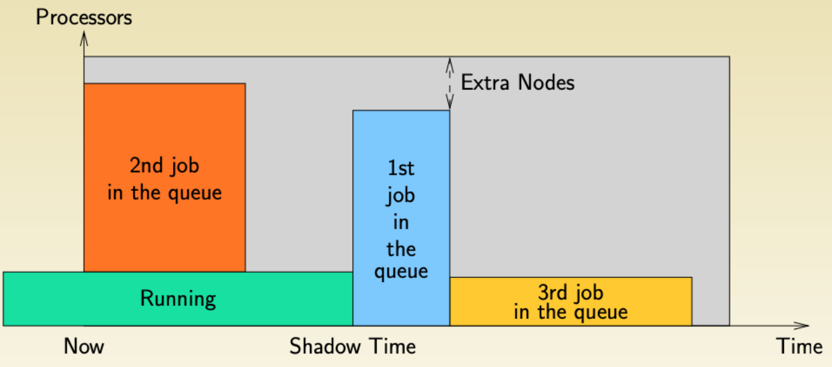
First job in the queue is never delayed by backfilled jobs

EASY



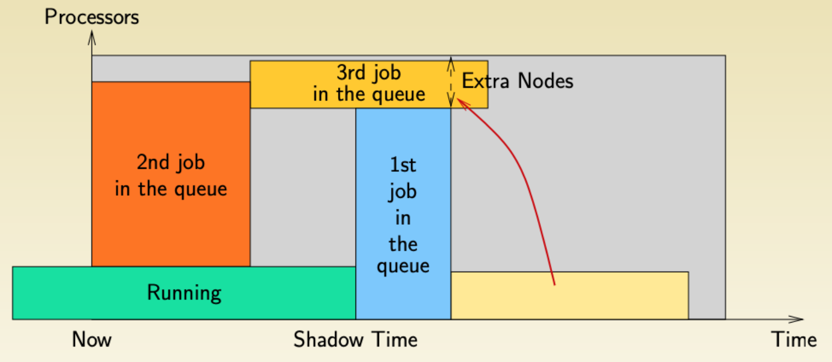
First job in the queue is never delayed by backfilled jobs

EASY



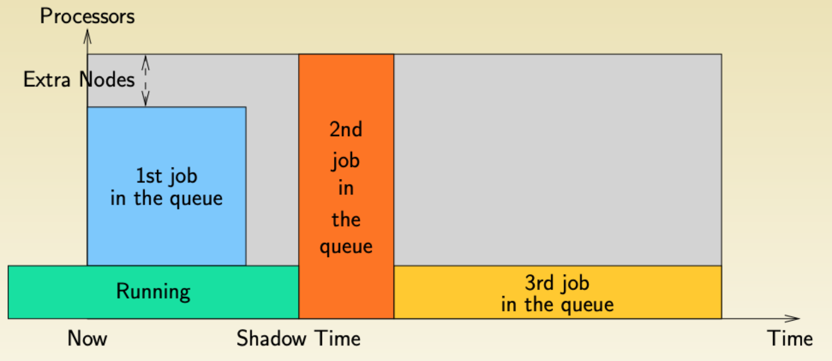
First job in the queue is never delayed by backfilled jobs

EASY



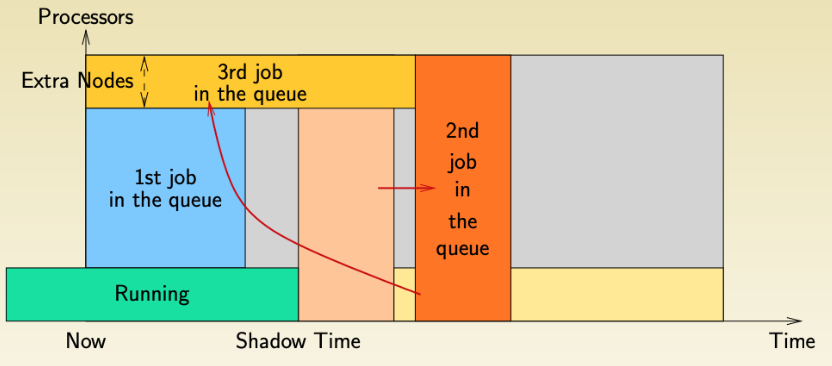
First job in the queue is never delayed by backfilled jobs

EASY



First job in the queue is never delayed by backfilled jobs
BUT other jobs may be delayed indefinitely!

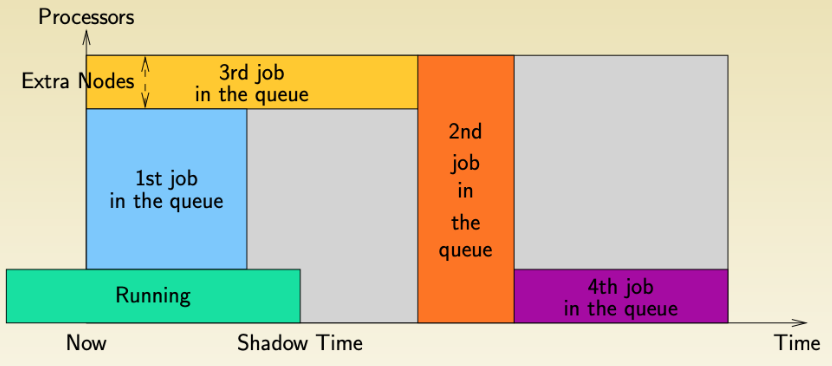
EASY



First job in the queue is never delayed by backfilled jobs

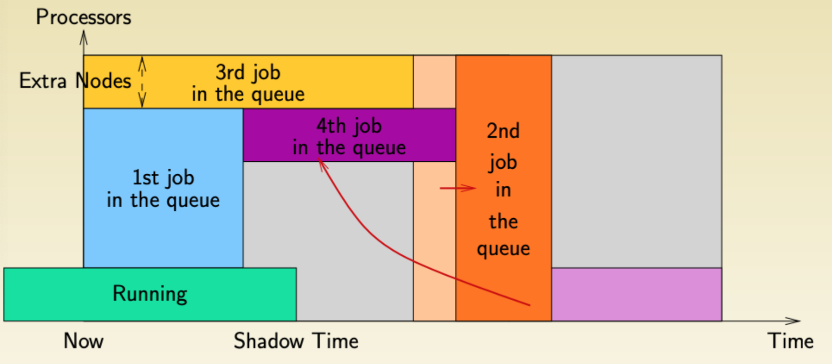
BUT other jobs may be delayed indefinitely!

EASY



First job in the queue is never delayed by backfilled jobs
BUT other jobs may be delayed indefinitely!

EASY



First job in the queue is never delayed by backfilled jobs

BUT other jobs may be delayed indefinitely!

EASY properties

Unbounded delay

- First job in the queue never delayed by backfilled jobs
- BUT other jobs may be delayed indefinitely!

No starvation

- Delay of first job in the queue is bounded by runtime of current jobs
- When first job completes, second job becomes first job in the queue
- Once it is the first job, it cannot be delayed further

Behavior

- EASY favors small long jobs and delays large short jobs

Conservative backfilling

Find holes in the schedule

- *Each* job has a *reservation time*
- A job may be backfilled
  only if it does not delay any other job ahead of it in the queue
- Fixes EASY unbounded delay problem
- More complicated to implement ☹

When does backfilling happen?

Possibly when

- A new job is released
- The first job in the queue starts execution
- When a job finishes early

A job is killed if it goes over

Users provide job runtime estimates
Trade-off: provide

- a tight estimate: you go through the queue faster (may be backfilled)
- a loose estimate: your job will not be killed

When does backfilling happen?

Possibly when

- A new job is released
- The first job in the queue starts execution
- When a job finishes early

Tricks

- Pick the right "shape" so that you'll be backfilled
- Chop up your job into multiple pieces
- Aggressively submit versions of the same job (different shapes), perhaps to multiple systems, and cancel when one begins
- ...

- a loose estimate: your job will not be killed

What's a good batch schedule?

- Define a metric of goodness for this on-line scheduling problem
- **Wait time**: time spent in the queue
  - Wait time is annoying, so likely a good thing to minimize
  - Not a great idea:
    - Job #1 needs 100h on 1000 nodes and waits 1h
    - Job #2 needs 1s on 1 node and waits 1h
    - Clearly, Job #1 is really happy ☺, and Job #2 is not happy at all ☹
- **Turn-around time**: Wait time + Execution time
  - Called *flow time* in scheduling literature
  - Not a great idea:
    - Job #1 needs 1h of compute time and waits 1s
    - Job #2 needs 1s of compute time and waits 1h
    - Clearly, Job #1 is really happy ☺, and Job #2 is not happy at all ☹

What's a good batch schedule?

- Define a metric of goodness for this on-line scheduling problem
- **Wait time**: time spent in the queue
  - Wait time is annoying, so likely a good thing to minimize
  - Not a great idea:
    - Job #1 needs 100h on 1000 nodes and waits 1h
    - Job #2 needs 1s on 1 node and waits 1h
    - Clearly, Job #1 is really happy ☺, and Job #2 is not happy at all ☹
- **Turn-around time**: Wait time + Execution time
  - Called *flow time* in scheduling literature
  - Not a great idea:
    - Job #1 needs 1h of compute time and waits 1s
    - Job #2 needs 1s of compute time and waits 1h
    - Clearly, Job #1 is really happy ☺, and Job #2 is not happy at all ☹

What's a good batch schedule?

- Define a metric of goodness for this on-line scheduling problem
- **Wait time**: time spent in the queue
  - Wait time is annoying, so likely a good thing to minimize
  - Not a great idea:
    - Job #1 needs 100h on 1000 nodes and waits 1h
    - Job #2 needs 1s on 1 node and waits 1h
    - Clearly, Job #1 is really happy ☺, and Job #2 is not happy at all ☹
- **Turn-around time**: Wait time + Execution time
  - Called *flow time* in scheduling literature
  - Not a great idea:
    - Job #1 needs 1h of compute time and waits 1s
    - Job #2 needs 1s of compute time and waits 1h
    - Clearly, Job #1 is really happy ☺, and Job #2 is not happy at all ☹

What's a good batch schedule?

- We want a metric that represents "happiness" for small, large, short, long jobs ☺ ☺ ☺ ☺
- **Slowdown**: (Wait time + Execution time) / Execution time
  - Called *stretch* in scheduling literature
  - Quantifies loss of performance due to competition for the processors
  - Takes care of the short vs. long job problem
  - Doesn't really say anything about job size ...
- Two possible objectives:
  - Minimize the **Sum** *stretch* (make jobs happy on average)
  - Minimize the **Max** *stretch* (make the least happy job as happy as possible)

What's a good batch schedule?

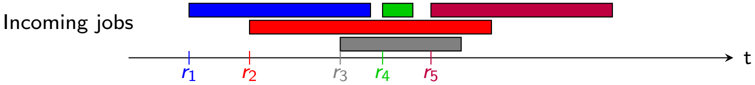
- We want a metric that represents "happiness" for small, large, short, long jobs

> **Flow time** measures the time that a job is in the system regardless of the service it requests; **the stretch measure** relies on the intuition that a job that requires a long service time must be prepared to wait longer than jobs that require small service times.
> *M. Bender et al, J. of Scheduling, 2004*

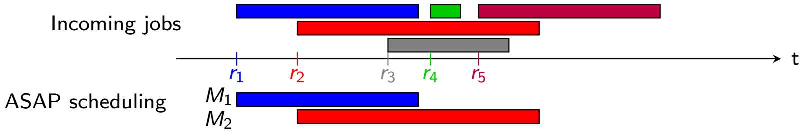
- Doesn't really say anything about job size ...

- Two possible objectives:
  - Minimize the **Sum** *stretch* (make jobs happy on average)
  - Minimize the **Max** *stretch* (make the least happy job as happy as possible)

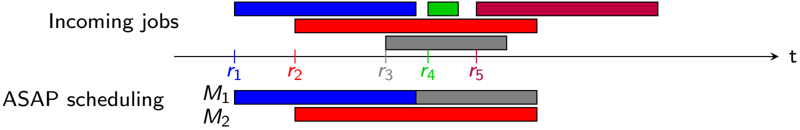
Minimizing maximum stretch



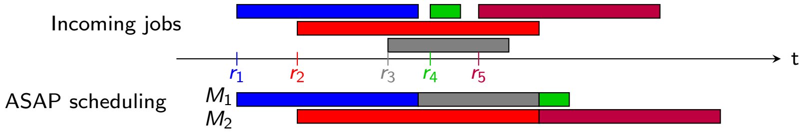
Minimizing maximum stretch



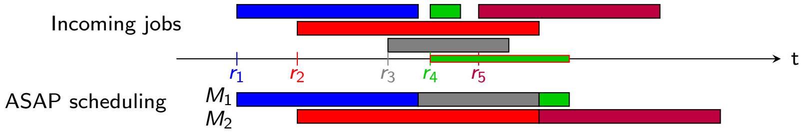
Minimizing maximum stretch



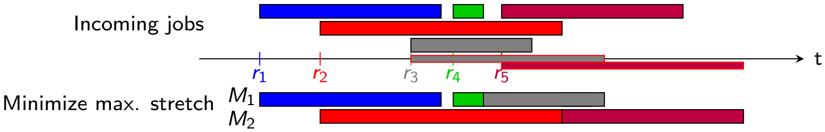
Minimizing maximum stretch



Minimizing maximum stretch



Minimizing maximum stretch



Online max stretch: Difficult

- The offline scheduling problem is NP-complete
- On one processor, with preemption allowed, there is a $O(\sqrt{X})$-competitive algorithm
  - X is the ratio of largest to smallest job duration
  - Competitive ratio: ratio to performance of an adversary who knows all jobs
- Without preemption, no approximation algorithm

A massive divide

Practice
- No preemption in batch scheduling
- Need for many scheduling configuration knobs

Theory
- Without preemption, we cannot do anything guaranteed anyway

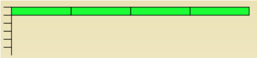
- The two remain very divorced
- Stretch used as a metric to evaluate how good scheduling is in practice
- Often, it is not the objective of the batch scheduler
- That objective is complex, sometimes mysterious, and not necessarily theoretically-motivated
- Bottom-line: Users hate the batch queue, and will use ingenuity to get ahead

Scheduling objectives

- User-oriented, performance
  - **Wait time** – Amount of time spent waiting before execution
  - **Turnaround time/Response time/Flow** – Amount of time between job release and completion
  - **Slowdown/Stretch** – Slowdown factor relative to time it would take on an unloaded system
- User-oriented, other criteria
  - **Cost** – Money paid for reservation
  - **Energy** – Energy consumed by job
- Platform-oriented
  - **Utilization** – Proportion of time spent doing computation
  - **Goodput** – Proportion of time spent doing successful computation
  - **Failure rate** – Proportion of interrupted jobs
  - **Total power** – Minimize power peak
  - **Carbon emission** – Minimize carbon emission (if green power sources available)

Outline

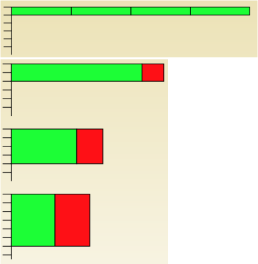
Flexible workloads: Parallel jobs



- **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

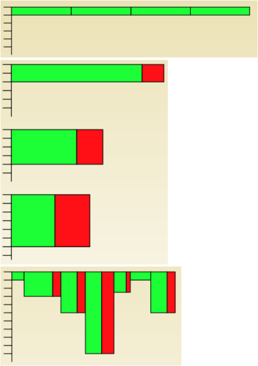
Flexible workloads: Parallel jobs

- **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



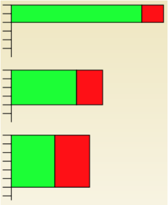
Flexible workloads: Parallel jobs

- **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



Flexible workloads: 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

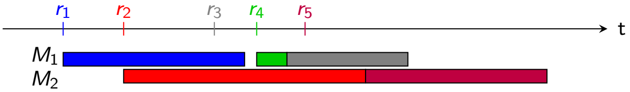
- Many checkpointable jobs are moldable
- These jobs are able to restart with a different allocation (size and shape)

⚠ Resizing impacts performance

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

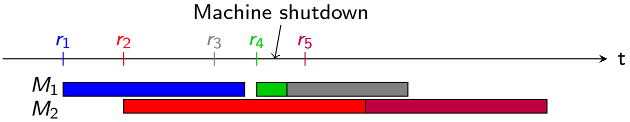
Scheduling techniques: Risk-aware?

1. **Which machine to shutdown?**



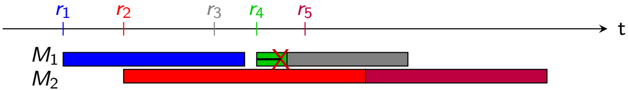
Scheduling techniques: Risk-aware?

1. **Which machine to shutdown?**



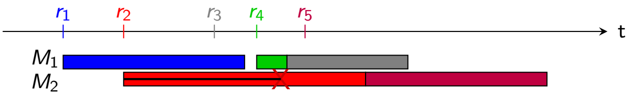
Scheduling techniques: Risk-aware?

1. **Which machine to shutdown?**



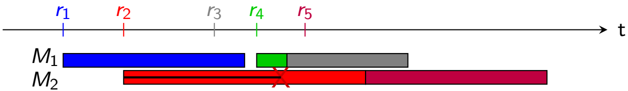
Scheduling techniques: Risk-aware?

1. **Which machine to shutdown?**



Scheduling techniques: 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?

Outline

Framework

- **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

- Number of alive machines evolves over time
  (either random-length phases, or fixed-length periods)
- Number of alive machines in the next phase/period 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 $\Rightarrow$ no flexibility in size
- Identical multicore machines
- Power consumption at time t proportional to $M_{alive}(t)$ (actual load not accounted for)
- No checkpoints
- Resource variation not known until change (no predictions)

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

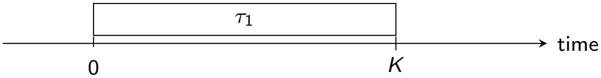
- Job τ_1 of size $c_1 = 1$ and duration $w_1 = K$ released at time $t = r_1 = 0$; Goodput of the machine at time $T = K$?

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 $T = K$?



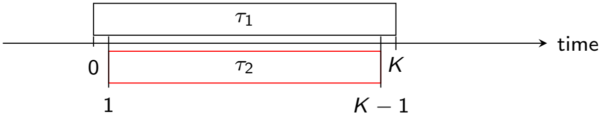
- Start τ_1 at time $s_1 > 0$: machine interrupted at time K

Complexity

Theorem

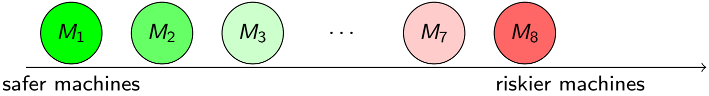
An adversary can force any schedule to achieve no goodput at all, even with a single unicore machine

- 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 $T = K$?



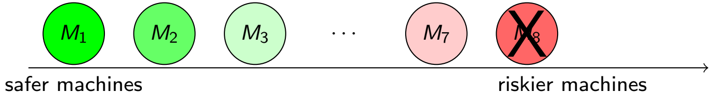
- Start τ_1 at time $s_1 = 0$: new job τ_2, machine interrupted at time $K-1$

Risk-aware



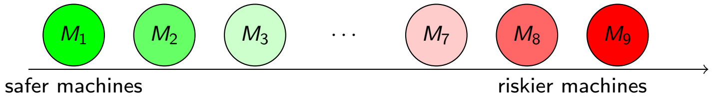
Risk-aware job allocation strategies

Risk-aware



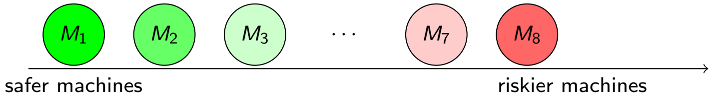
Risk-aware job allocation strategies

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

vith

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

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

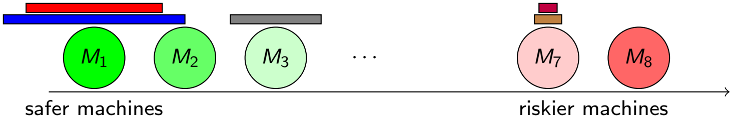
FirstFitUnaware: Shutdown random machines whenever necessary

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

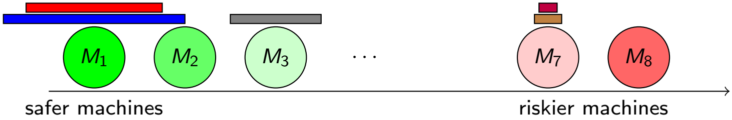
TargetStretch



- 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

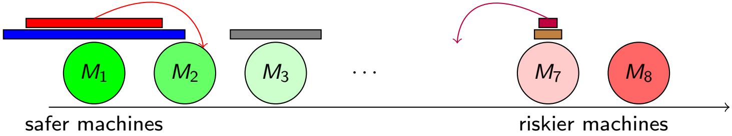
TargetASAP & PackedTargetASAP

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



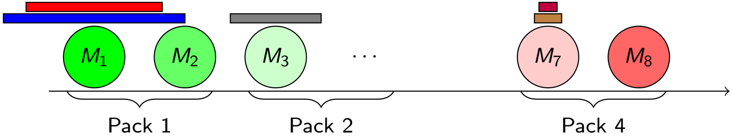
TargetASAP & PackedTargetASAP

- 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

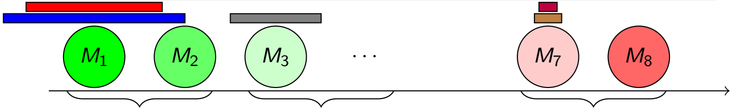
- TargetStretch: potential bad utilization
  No flexibility for mapping to another free machine
- TargetASAP:
  - Start job immediately on target machine or closest machine in neighborhood
  - If not possible, assign on target machine if target stretch not exceeded
  - Otherwise, assign on machine where 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

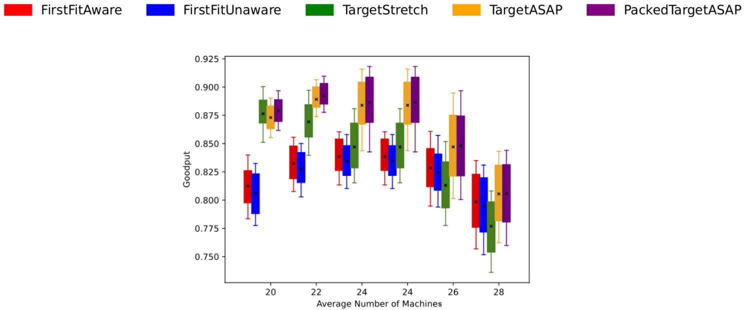
In-house simulator, using a combination of two traces:

- 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: $\phi = 20$ minutes
- 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

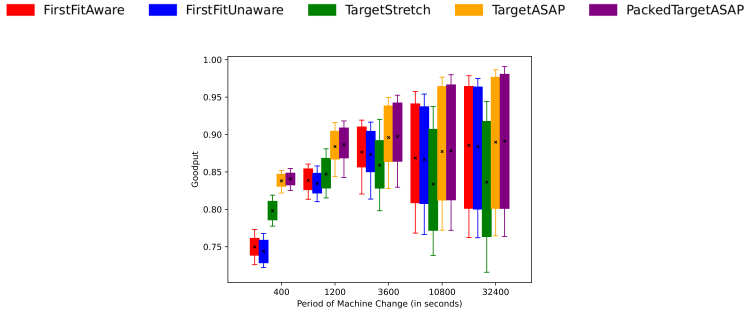
Varying the number of machines



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

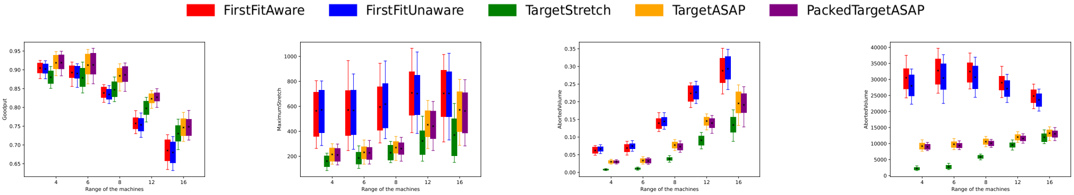
Varying the period of machine variation



BORG

- With low period (many changes), TARGETSTRETCH better by preserving long jobs
- Goodput increases with period: less changes $\Rightarrow$ 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

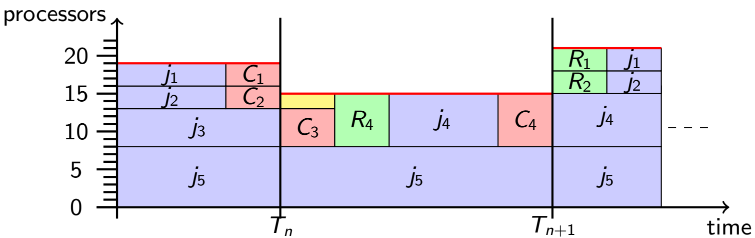
Problem: Schedule parallel rigid jobs with a variable number of processors

Hypotheses:

- A job can be checkpointed and recovered
- Knowledge of the duration of each *section* before a change in available processors, and bound on #proc difference ⇒ Focus on a single section

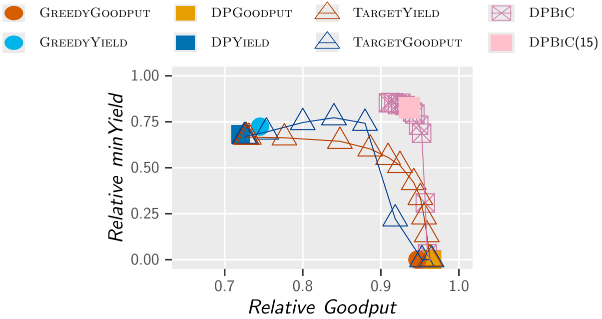
Additional constraint:

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



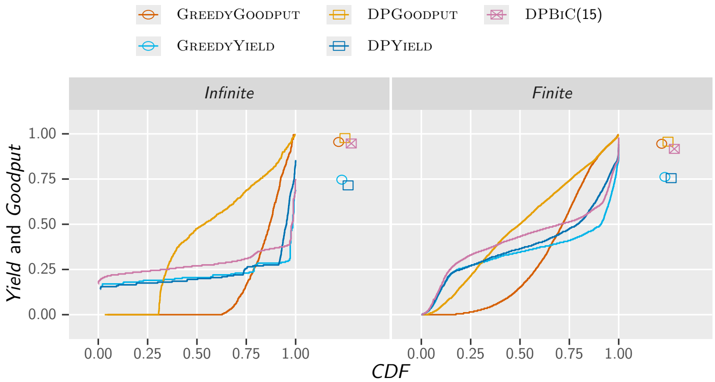
Algorithms and evaluation

- Sophisticated dynamic programming algorithms to optimize goodput and/or yield at the end of a section, assuming jobs are infinite
- Evaluation on job traces, with infinite and finite jobs



- Improvement of novel strategies over greedy approaches:
  Great trade-off of bi-criteria DP algorithm, results close to upper bounds

Comparing infinite vs finite settings



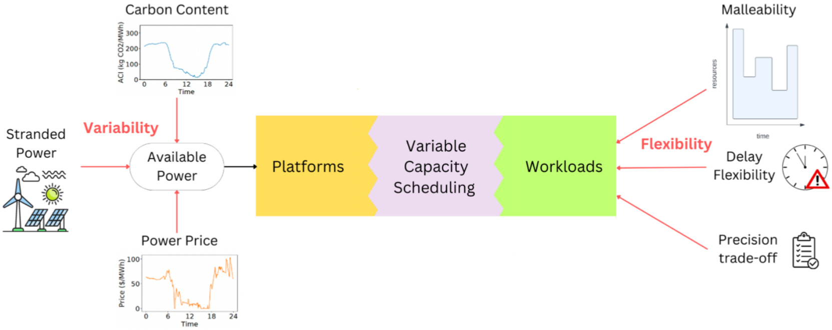
- Algorithms assuming infinite jobs, adapted to finite jobs
- Similar behavior in the simulations: identical goodput and same tendencies for yield when looking at the CDF (min yield always 0 for finite jobs)

Summary

- Formal model of a scheduling problem with variable number of processors
- Design of multiple dynamic programming algorithms, building on the case of infinite jobs
- Adaptation of algorithms for finite jobs
- Simulations that give results close to the maximum bounds, both for infinite and finite jobs

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, `https://inria.hal.science/hal-04159509v1`
Second workshop planned in September 2025

Take-aways

- Today's case study without checkpoints: restricted instance ☹
  Risk-Aware Scheduling Algorithms for Variable Capacity Resources;
  PMBS workshop at SC'23
- With checkpoints: Many assumptions – exact knowledge of period changes, bound on machine variation

- In some particular settings, possible to design clever ad-hoc solutions
- Instantiate problems from real case studies, good performance

Future research directions

- Relax some hypothesis to explore further problems
- Explore other kinds of jobs, in particular moldable jobs, to further exploit workload flexibility
- Study other objective functions (carbon emissions, memory, etc.)
- Take into account energy cost of communications
- In the longer term, come up with new models for resource variability, and multi-criteria metrics accounting for performance and sustainability
- The world does not have infinite resources: limit IT growth, help users accept new constraints (importance of fairness)
  $\Rightarrow$ Societal impact of variable capacity scheduling

Thanks to my colleagues Henri Casanova, Arnaud Legrand, and Yves Robert, from whom I borrowed some slides ☺