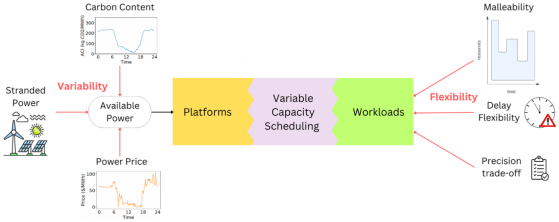
Scheduling for Variable Capacity Resources



Yves Robert
Professor, Ecole Normale Supérieure de Lyon
Visiting Scientist, University of Tennessee Knoxville

Co-Authors


ENS Lyon: Anne Benoit, Lucas Perotin

Univ. Tennessee Knoxville
Thomas Herault



U. Chicago
Andrew A. Chien, Rajini Wijayawardana, Chaojie Zhang

Slides Borrowed From friends

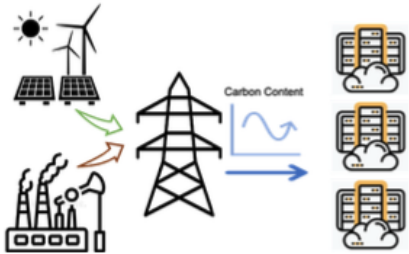
Univ. Hawaii
Henri Casanova

CNRS Grenoble
Arnaud Legrand

Outline

Outline

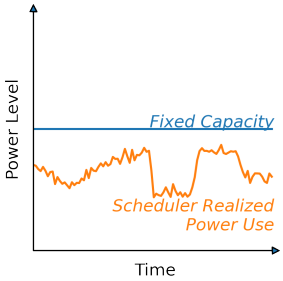
Variable Power



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

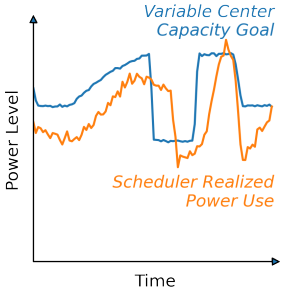
$\Rightarrow$ Variable power

Scenarios For HPC / Data Centers: TODAY



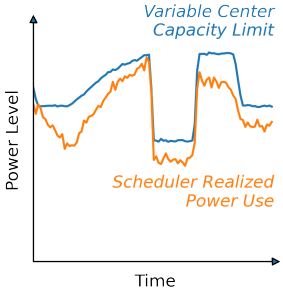
- Fixed, contractual envelope for power
- Datacenter management & resource management indifferent to external conditions
- Scheduler decisions affect power use, well below maximum Thermal Design Power

Scenarios For HPC / Data Centers: MATCHING



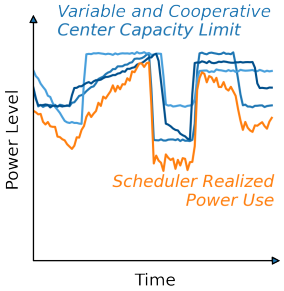
- Recognizes the changing external power price and carbon emissions
- Seeks to change scheduling decisions to reduce power costs or carbon emissions
- Running "power hog" jobs when power is cheap or renewable content is high

Scenarios For HPC / Data Centers: FLEXIBLE



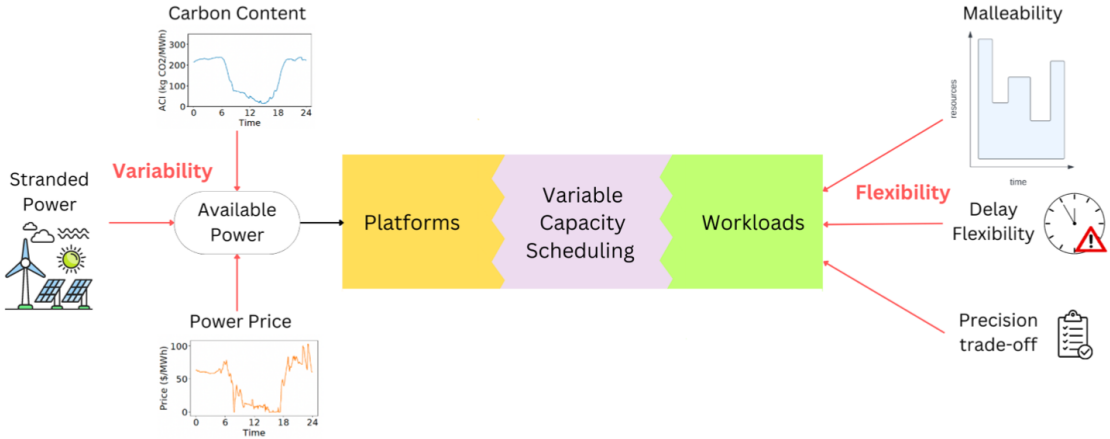
- Variable power level enforced on center by external environment (local power controller to manage power cost or carbon, power grid for grid stability)
- Operates within specified variable power envelope (and corresponding capacity) as a constraint that bounds resource management
- Similar to power cap, but now variable over time and dictated externally

Scenarios For HPC / Data Centers: COOPERATIVE

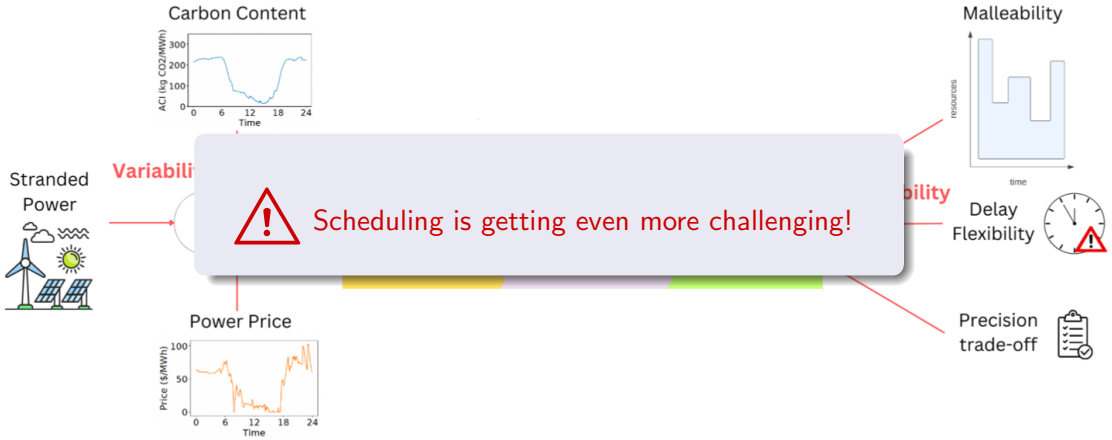


- Both external circumstance and internal operational constraints (e.g. workload)
- Negotiates a variable capacity envelope with external controller
- This negotiation allows the needs to be balanced:
  could maintain operations during a power crisis, or save the grid from blackout

Big Picture


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

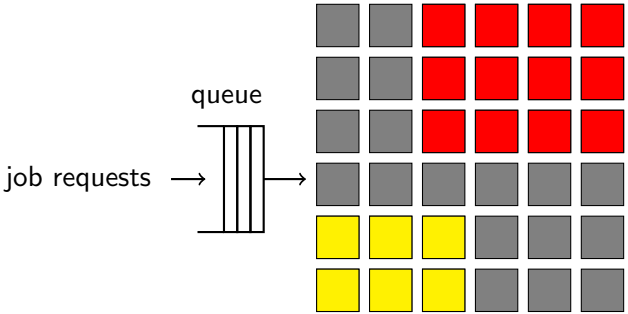
Big Picture


Carbon Content
ACI (kg CO2/MWh)
300
200
100
0
0 6 12 18 24
Time
Malleability
resources
time
Stranded Power
Variabili
bility
Scheduling is getting even more challenging!
Delay Flexibility
Precision trade-off
Power Price
Price ($/MWh)
100
50
0
0 6 12 18 24
Time

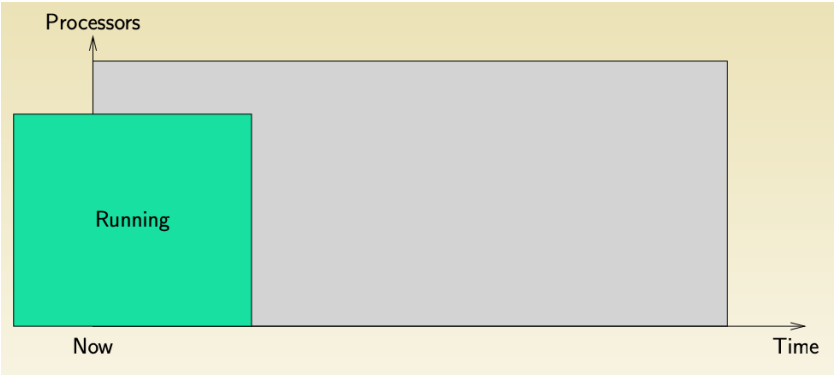
Outline

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, a.k.a. *RJMS*, is responsible for the sharing

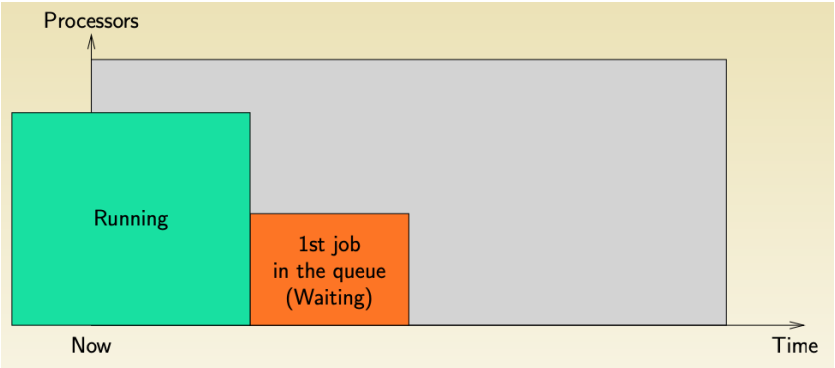


General Principle



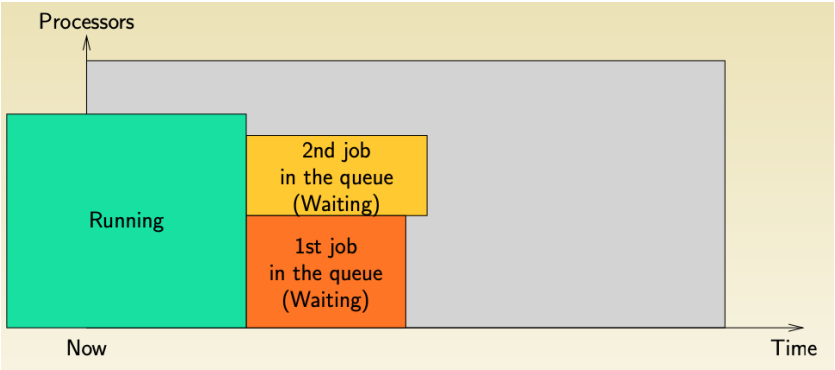
Jobs are released one after the other and are scheduled upon arrival

General Principle



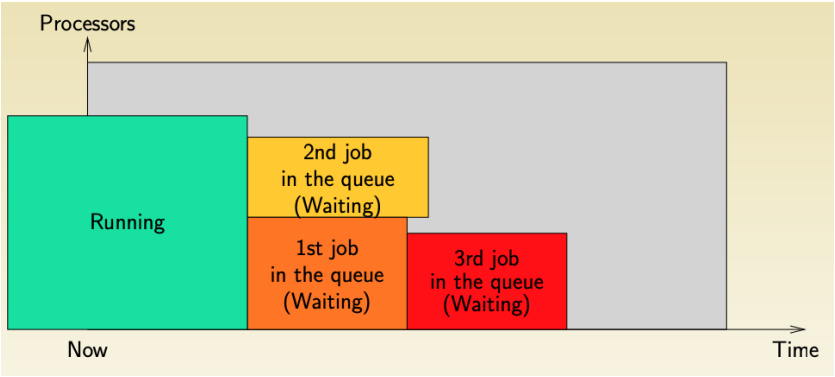
Jobs are released one after the other and are scheduled upon arrival

General Principle



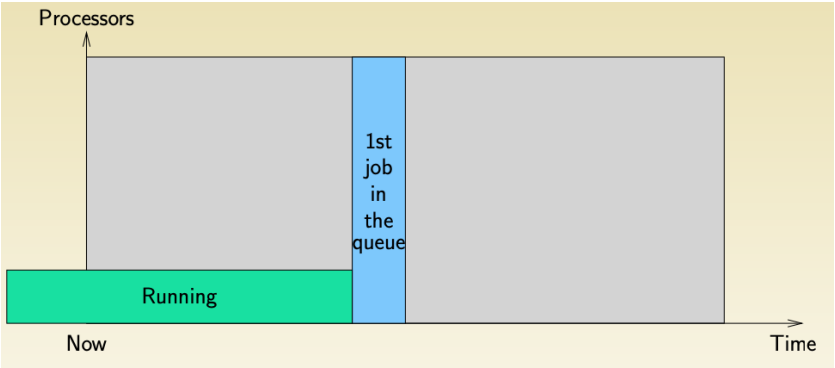
Jobs are released one after the other and are scheduled upon arrival

General Principle



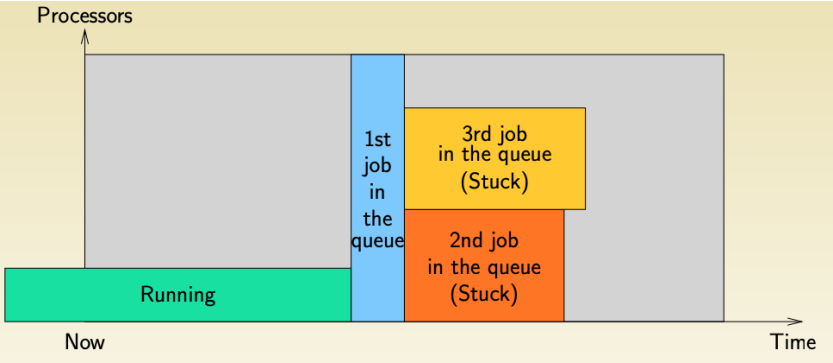
Jobs are released one after the other and are scheduled upon arrival

Backfilling



FCFS + FirstFit = simplest scheduling strategy

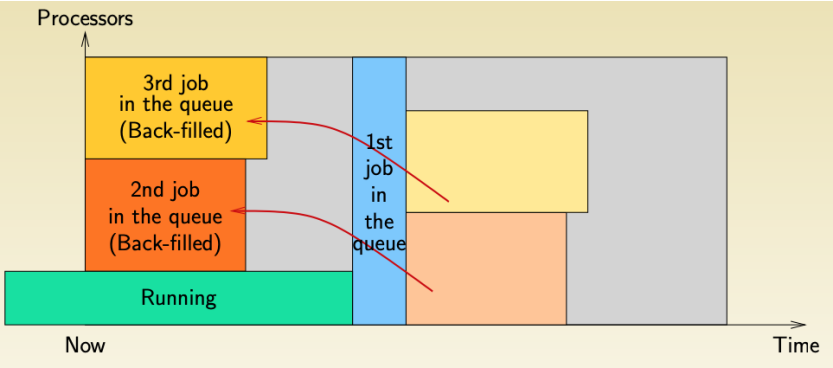
Backfilling



FCFS + FirstFit = simplest scheduling strategy

Fragmentation ☹

Backfilling



FCFS + FirstFit = simplest scheduling strategy

Fragmentation ☹ ⇒ need for backfilling

EASY Backfilling

Extensible Argonne Scheduling System

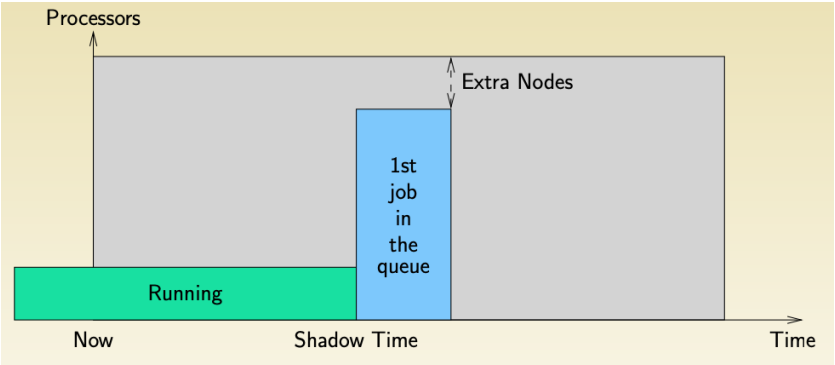
Maintain only one *reservation time*, for first job in the queue

Shadow time starting execution of first job in the queue

Extra nodes number of nodes idle at shadow time

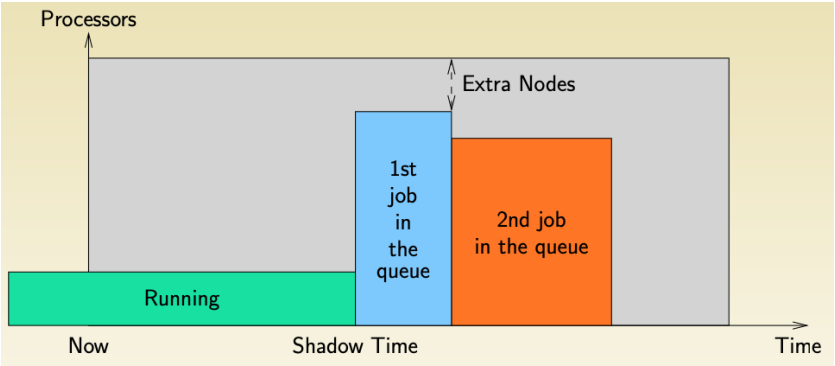
1. Go through the queue in order, starting with second job
2. Backfill a job
   - either if it will terminate by shadow time
   - or it needs no more nodes than the extra nodes

EASY



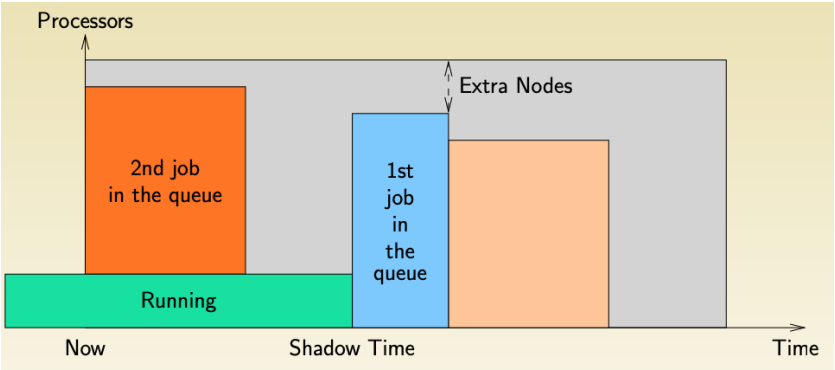
First job in the queue is never delayed by backfilled jobs

EASY



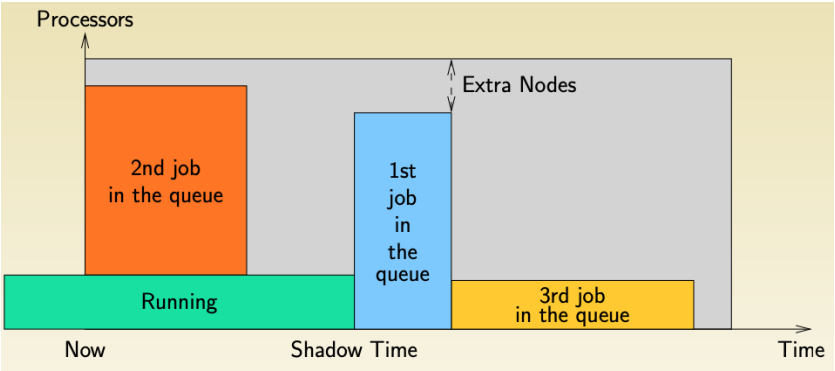
First job in the queue is never delayed by backfilled jobs

EASY



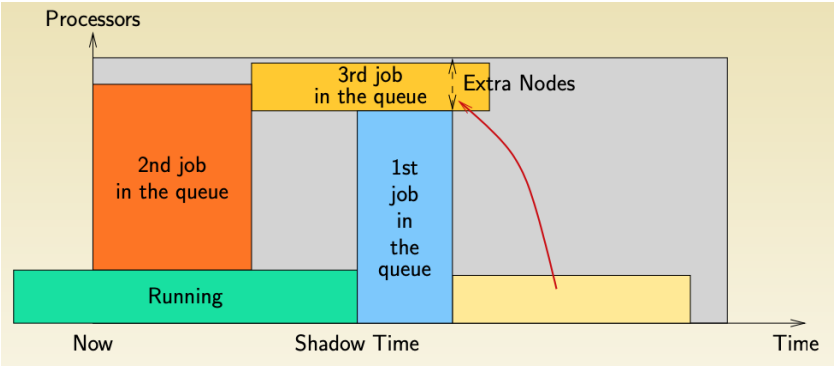
First job in the queue is never delayed by backfilled jobs

EASY



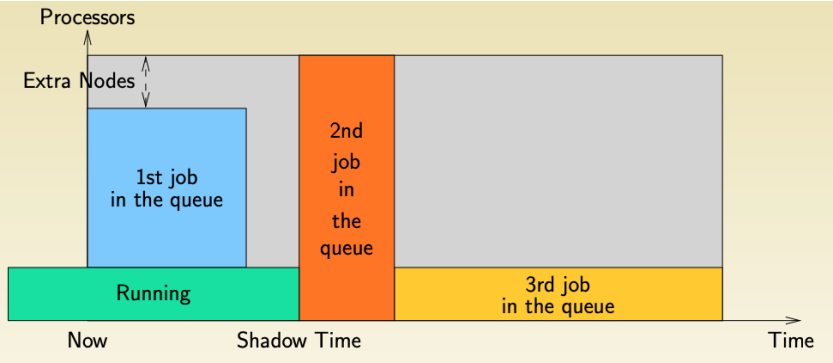
First job in the queue is never delayed by backfilled jobs

EASY



First job in the queue is never delayed by backfilled jobs

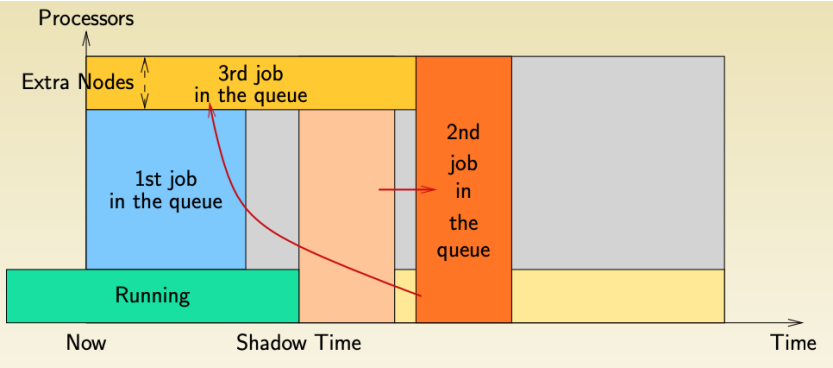
EASY



First job in the queue is never delayed by backfilled jobs

BUT other jobs may be delayed indefinitely!

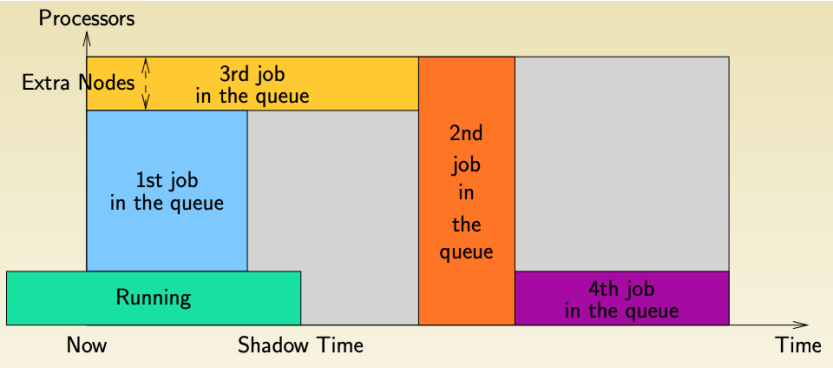
EASY



First job in the queue is never delayed by backfilled jobs

BUT other jobs may be delayed indefinitely!

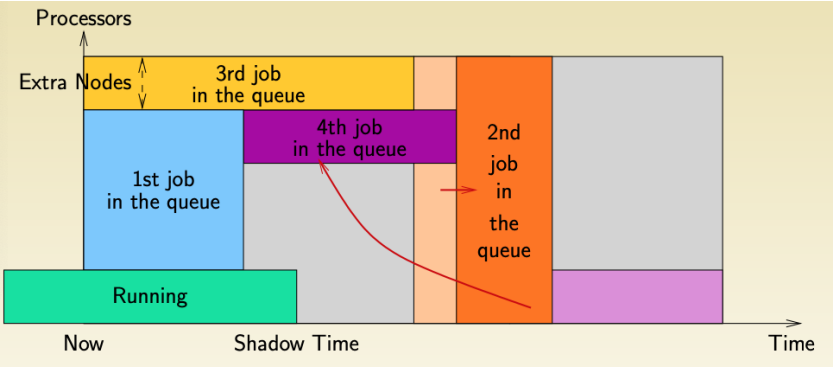
EASY



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 is 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
- [illegible]

> 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)

What's A Good Batch Schedule?

- We want a metric that represents "happiness" for small, large, short, long jobs
- [illegible]
  - Doesn't really say anything about job size ...
- Distributive justice;

$$\text{Stretch} = \frac{\text{Completion time} - \text{Submission time}}{\text{Processing time}}$$

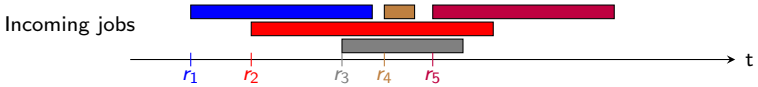
$$\text{Justice} = \frac{\text{Reward} - \text{Cost}}{\text{Investment}}$$

V.G. Pinheiro, PhD thesis, Universidade São Paulo, 2014

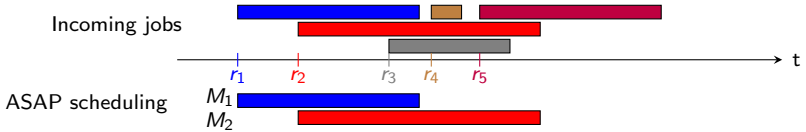
Flow time measures the time that a job is in the system regardless of the service it requests; the stretch measure relies on the intuition that a job that requires a long service time must be prepared to wait longer than jobs that require small service times.
M. Bender et al, J. of Scheduling, 2004

[illegible]sible)

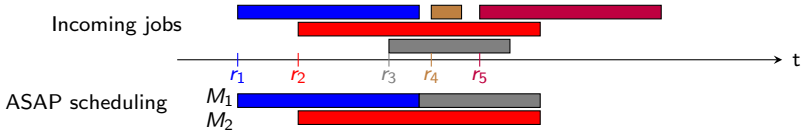
Minimizing Maximum Stretch



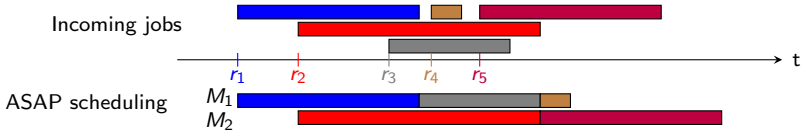
Minimizing Maximum Stretch



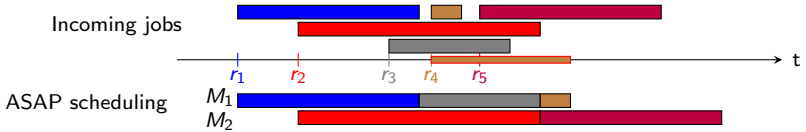
Minimizing Maximum Stretch



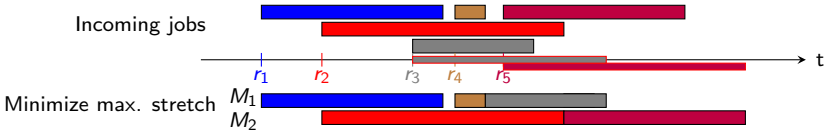
Minimizing Maximum Stretch



Minimizing Maximum Stretch



Minimizing Maximum Stretch



Online Max Stretch: Difficult

- The offline scheduling problem is NP-complete
- On 1 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 can't do anything guaranteed anyway

- The two remain very divorced
- Stretch used as a metric to evaluate how good scheduling is in practice
- Often isn't 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)

Reservation Cost

For a job with a reservation of length R and an actual execution of length W, pay

$$\alpha R + \beta \min(W, R) + \gamma$$

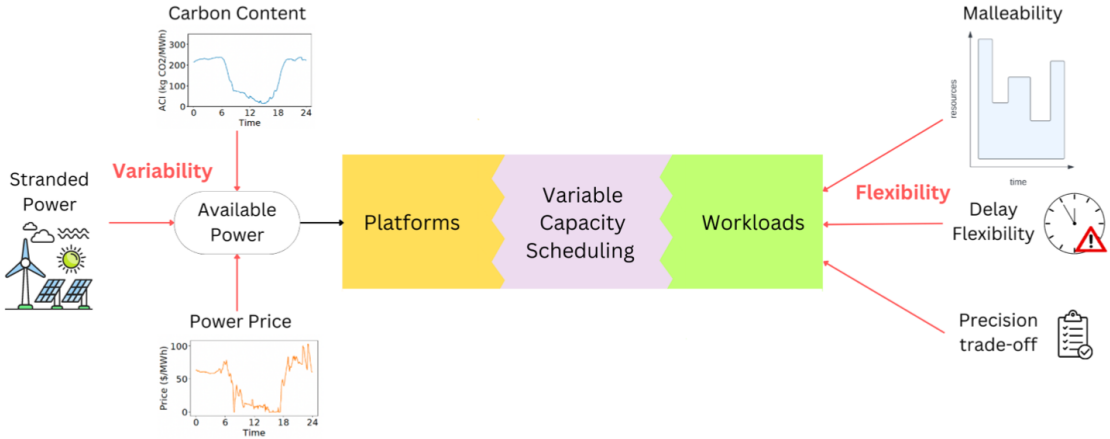
- **HPC**
  Users request a set of resources for a given number of hours
  Only pay for the hours actually spent
  Assigned waiting queue, hence wait time, both depend upon reservation length
- **Cloud**
  Complicated price mechanisms
  E.G.: Amazon AWS Reserved Instances (RI) discounted (up to 75%) compared to on-demand pricing

Cloud Scheduling: A Wide Spectrum Of Techniques

- More constraints: QoS, SLA, priorities, ...
- Several approaches: on-demand, advanced reservation, preemption/migration, ...
- Resource selection and mapping, much harder due to heterogeneity
- VM sharing, scheduling at VM level
- Many heuristics: FCFS, FirstFit, BestFit, SJF, Max-Min, ...
- Many objectives and multi-criteria trade-offs

Outline

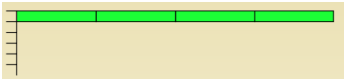
Big Picture


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

Outline

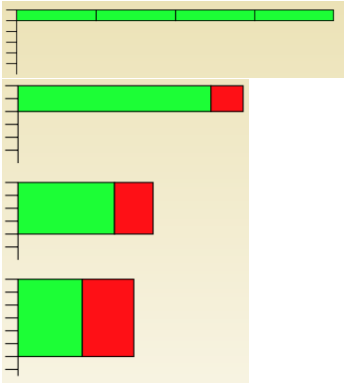
Parallel Jobs (1/2)

- Rigid jobs
- Moldable jobs
- Malleable jobs



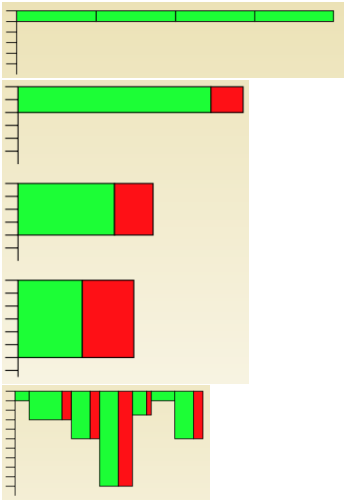
Parallel Jobs (1/2)

- Rigid jobs
- Moldable jobs
- Malleable jobs



Parallel Jobs (1/2)

- Rigid jobs
- Moldable jobs
- Malleable jobs



Parallel Jobs (2/2)

- **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

Speedup Models

- Roofline model: $t(p_j) = \frac{w}{\min(p_j, \bar{p}_j)}$, for some $1 \leq \bar{p}_j \leq P$
- Communication model: $t(p_j) = \frac{w}{p_j} + (p_j - 1)c$,
  where c is the communication overhead
- Amdahl's model: $t(p_j) = w\left(\frac{1-\gamma}{p_j} + \gamma\right)$,
  where γ is the inherently sequential fraction
- General model: $t_(p_j) = \frac{w_j(1-\gamma_j)}{\min(p_j, \bar{p}_j)} + w_j\gamma_j + (p_j - 1)c_j$,
  a combination of all models
- Arbitrary model: $t(p_j)$ is an arbitrary function of p_j, often with area $a(p_j) = p_j \times t(p_j)$ non-decreasing (no superlinear speed-up)

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

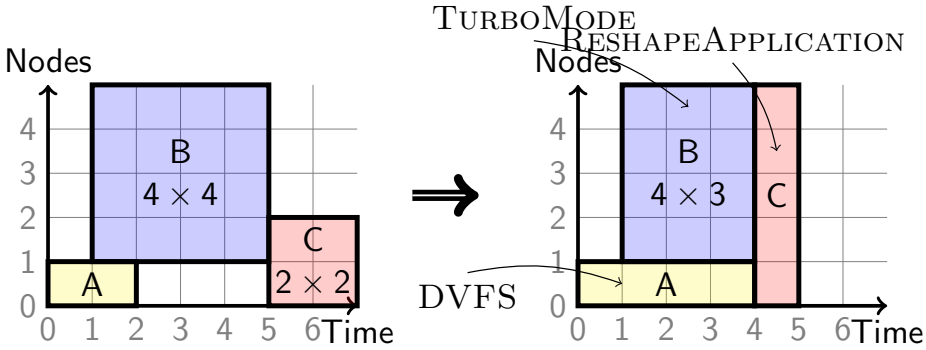
- Many checkpointable jobs are moldable
- These jobs are able to restart with a different allocation (size and shape)

Resizing impacts performance

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

Outline

Resizing Jobs: DVFS, TurboMode, ReshapeApplication



- Jobs ≡ rectangles: length = duration, and height = number of processors
- Positioning these rectangles ⇒ no overlap & optimize metric

DVFS

- Nominal frequency used by default
- Lower frequency: dynamic energy $\searrow$, time & static energy $\nearrow$
- Power at frequency f_j:

$$P(f_j) = P_{static} + P_{dyn}(f_j) = P_{static} + (P_{indep} + C \times f_j^3)$$

- Application-level $\rightarrow$ batch scheduler
- No real incentive as of today on HPC centers:
  longer reservation $\Rightarrow$ *higher price*

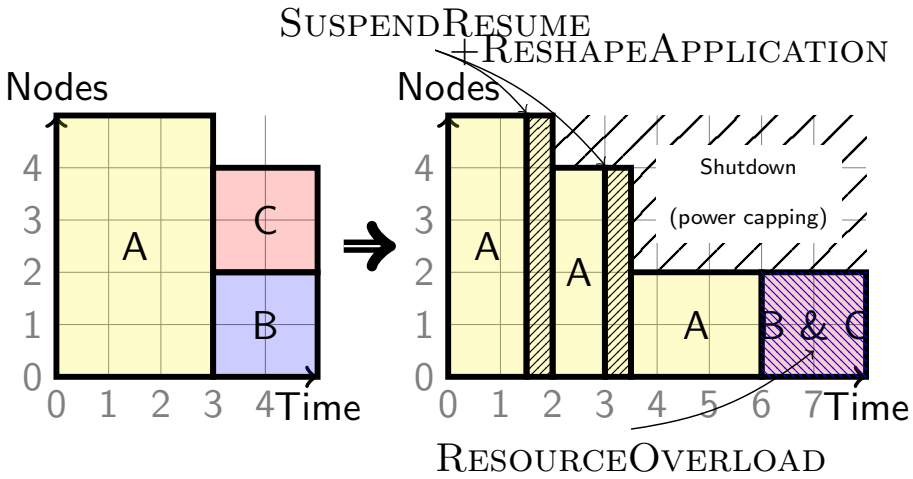
TurboMode

- Overclock with higher frequency than nominal
- Relevant when available resources going to be reduced soon:
  better speed up and terminate a long-running application
  before some of its resources become unavailable due to power capping
- Important for large/capacity applications (huge waste if failure & re-execution)
- ⚠ Increased heat dissipation $\Rightarrow$ negative long-term effect

ReshapeApplication

- Todays HPC centers
  - Job schedulers like IBM Spectrum LSF (Load Sharing Facility):
    the user specifies a range of resources (e.g., several grid sizes)
  - ReshapeApplication opens new flexibility in scheduling decisions,
    allows for optimizing resource usage from the platform point of view
    while reducing or preserving response time for the user
- Matching, Flexible-Power & Cooperative HPC centers
  - The user defines multiple alternative sets of constraints
    (#nodes, #cores, walltime, number of accelerators, etc...) ordered by preference
  - When shutting down resources. combine ReshapeApplication and
    SuspendResume to enable a dynamic reshape
    (need to update job parameters on the fly)

Reassigning Jobs: SuspendResume, ResourceOverload



SuspendResume

- Application-level checkpoint
  - Can restart in a different configuration (ReshapeApplication)
  - Take checkpoints only at specific application instants
- Transparent checkpoint (DMTCP)
  - Complete snapshot of process state
  - Can provide preemptive mechanism
  - Restore as is $\Rightarrow$ no reshaping
- HPC / Data centers
  - Todays HPC centers: schedule long-running applications by parts
  - Matching HPC centers: suspend high-energy applications for periods when the cost of energy is inflated
  - FlexiblePower and Cooperative HPC centers: necessary when resources are shut down just to meet variable power cap

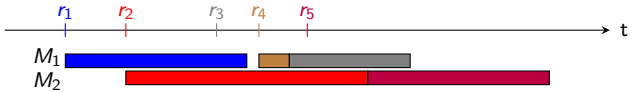
ResourceOverload

- Heavy memory constraints
- Higher flexibility level (virtualization)
- Needs incentive for the user

Outline

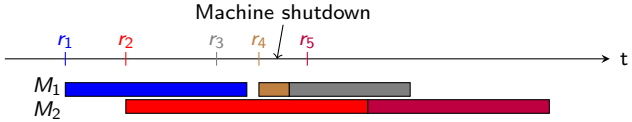
Risk Aware?

1. **Which Machine To Shutdown?**



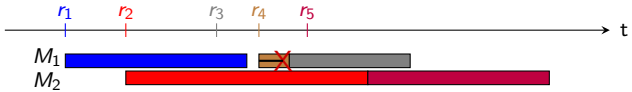
Risk Aware?

1. **Which Machine To Shutdown?**



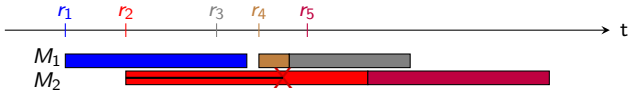
Risk Aware?

1. **Which Machine To Shutdown?**



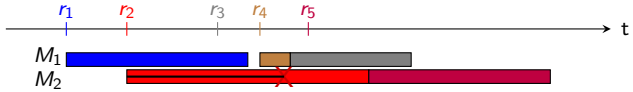
Risk Aware?

1. **Which Machine To Shutdown?**



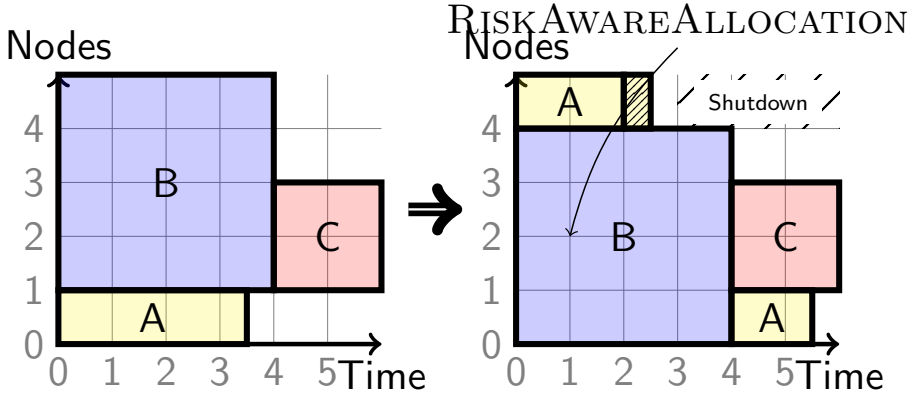
Risk Aware?

1. **Which Machine To Shutdown?**



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

Little Example



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?
- A

Scheduling opportunity & challenge

- Nodes ordered according to non-increasing 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

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 $M_{alive}(t)P \leq P(t)$

Rigids 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 $\Rightarrow$ no flexibility in size
- Identical multicore machines, and no DVFS
- No checkpoints $\Rightarrow$ no SuspendResume
- Power consumption at time t proportional to $M_{alive}(t)$ (actual load not accounted for)
- Resource variation not known until change

Alternative Approaches For Power Variation

1. Execution can continue at higher price when machine removed from the pool
   - Scenario with alive machines powered by green sources, brown power used as complement ☺
   - Requires *total cost* as a complementary objective, and a model to account for it ☹
2. Variations of capacity are known some time in advance
   - Jobs running on machines that are going to be switched off soon: $\Rightarrow$ can take a proactive checkpoint ☺
   - Requires many additional parameters:
     - prediction mechanism: prediction time, recall and precision
     - job characteristics: checkpointable or not, checkpoint duration

Alternative Approaches For Power Variation

1. Execution can continue at higher price when machine removed from the pool
   - Scenario with alive machines powered by green sources,

Using a simple Markov model
- Variations in wind power nicely obey such a Markov model
- Variations in solar power would require a more complicated model

(e.g., heterogeneous Markov chain to account for day or night)

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 $[0, T]$:

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

- $\textsc{Goodput}(T)$ – 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 $[0, T]$:

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

- $\textsc{Goodput}(T)$ – 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 (1/2)

Theorem

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

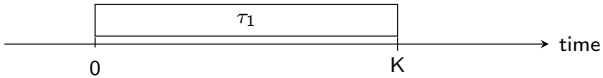
- Job τ_1 of size $c_1 = 1$ and duration $w_1 = K$ released at time $t = r_1 = 0$; Goodput of the machine at time $T = K$?

Complexity (1/2)

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 $T = K$?



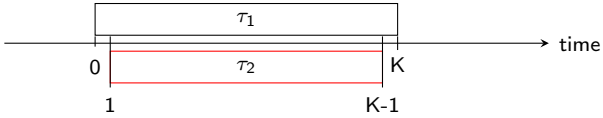
- Start τ_1 at time $s_1 > 0$: machine interrupted at time K

Complexity (1/2)

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 $K - 1$

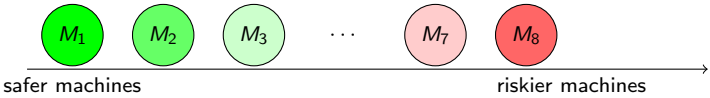
Complexity (2/2)

Theorem

Unless $P = NP$, there is no constant polynomial-time approximation algorithm of the makespan for the offline instance of the problem with parallel unicore machines

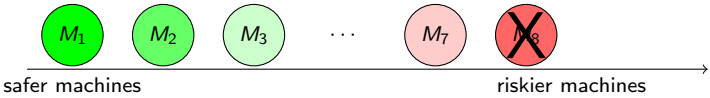
Outline

Risk-Aware



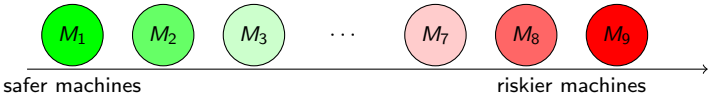
Risk-aware job allocation strategies

Risk-Aware



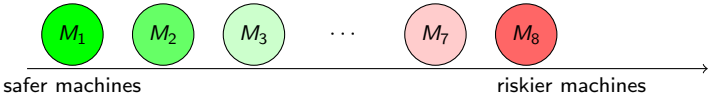
Risk-aware job allocation strategies

Risk-Aware



Risk-aware job allocation strategies

Events



- **Job Arrival**
  When a job is released, when 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

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

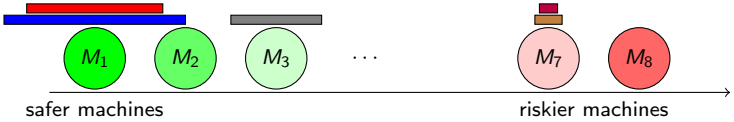
Risk-aware

- 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

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

TargetStretch

Be ready for some technicalities ... ☹

- **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

TargetStretch

Be ready for some technicalities ... ☹

Job arrival

- Assign each new job τ_i a category C_i according to its area
- Compute target machine number M_i^c proportionally in $[1, M_{use}]$ where M_{use} is current number of risk-free machines
- if job can start immediately or target stretch $S^+$is met, assign τ_i to M_i^c
- Otherwise, explore a neighborhood of M_i^c

$$C_i = \frac{\sum_{k \in \mathcal{J}', w_k \geq w_i} w_k c_k}{\sum_{k \in \mathcal{J}'} w_k c_k} \qquad M_i^c = \begin{cases} \lfloor C_i M_{use} \rfloor + 1 & \text{if } C_i < 1 \\ M_{use} & \text{otherwise} \end{cases}$$

TargetStretch

Be ready for some technicalities . . . ☹

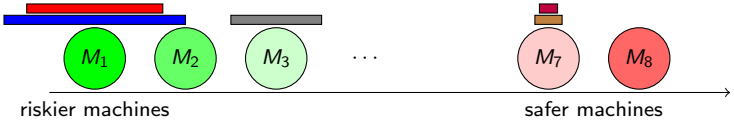
Machine Addition/Removal
Recompute system utilization and update M_{use} accordingly (initially $M_{use} = M^-$)
Re-allocate pending jobs (use global queue and sort by release time)

Consider neighboring machines if target stretch not achievable

- **Machine Addition/Removal**
  Set of risk-free machines recomputed
  Re-allocate pending jobs

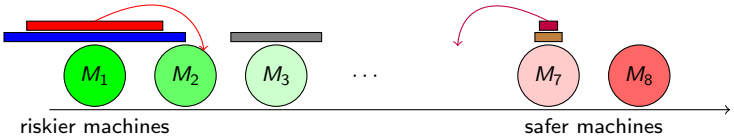
TargetASAP & PackedTargetASAP

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



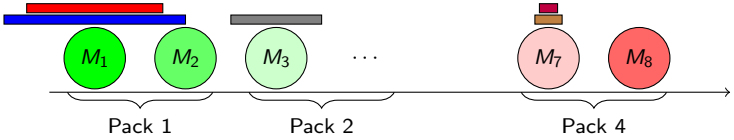
TargetASAP & PackedTargetASAP

- 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

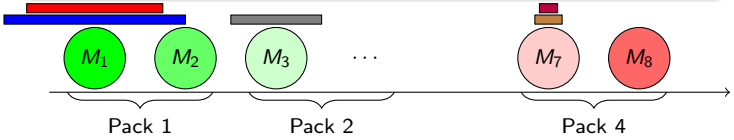
- TargetStretch: potential bad utilization
  No flexibility for mapping to another free machine
- TargetASAP:
  - Start job immediately on target machine or closest machine in neighborhood
  - If not possible, assign on target machine if target stretch not exceeded
  - Otherwise, assign on machine where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



Outline

Simulation Setting

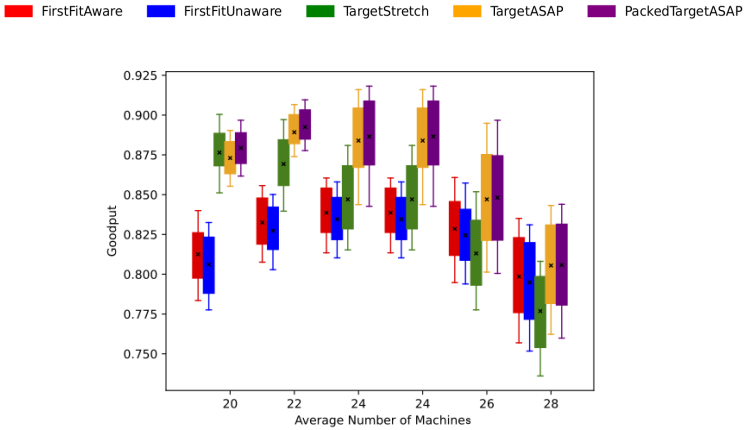
In-house simulator, using a combination of two traces:

- 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)

Dimensioning

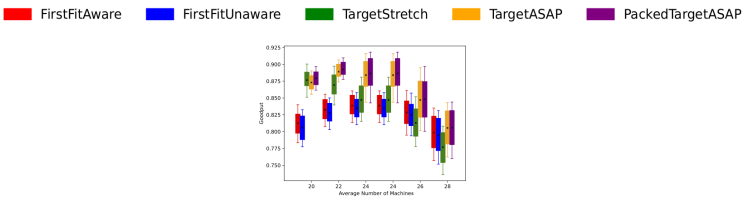
- Number of available machines always in $[M_{avg} - M_{ra}, M_{avg} + M_{ra}]$
- Total work hours $\approx$ maximum capacity of 26 machines each with 24 cores, running during 2 weeks with full peak load
- Average number of machines: $M_{avg} = 24$
- Period of machine variation: $\phi = 20mn$
- Range 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

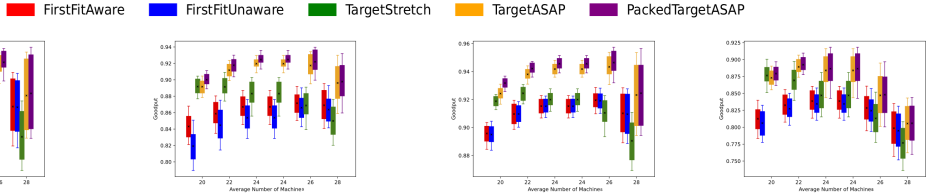
Varying The Number Of Machines



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

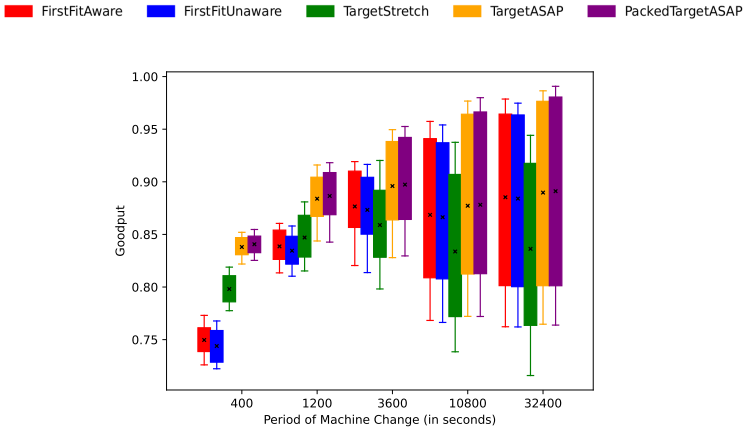
Varying The Number Of Machines



SyntheticUniform SyntheticLogScale Synthetic3Types Borg

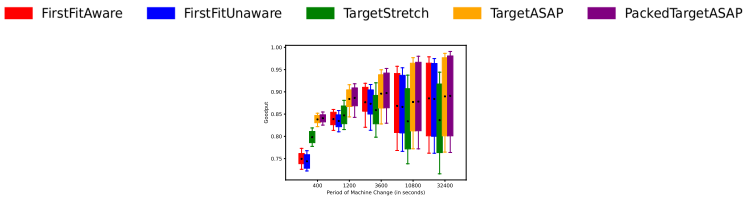
Limited impact of workflow type

Varying The Period Of Machine Variation



BORG

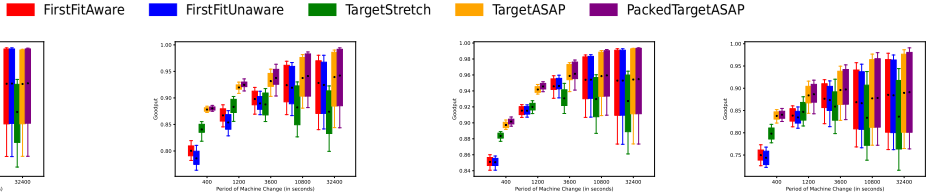
Varying The Period Of Machine Variation



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)

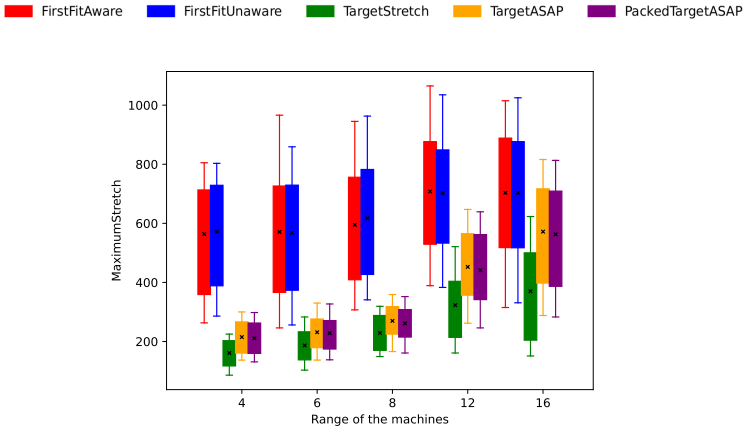
Varying The Period Of Machine Variation



SyntheticUniform SyntheticLogScale Synthetic3Types Borg

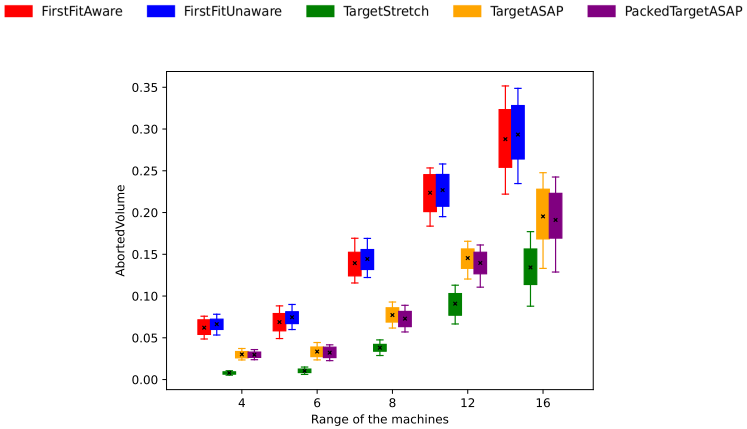
As before, limited impact of workflow type

Exploring Other Metrics For BORG



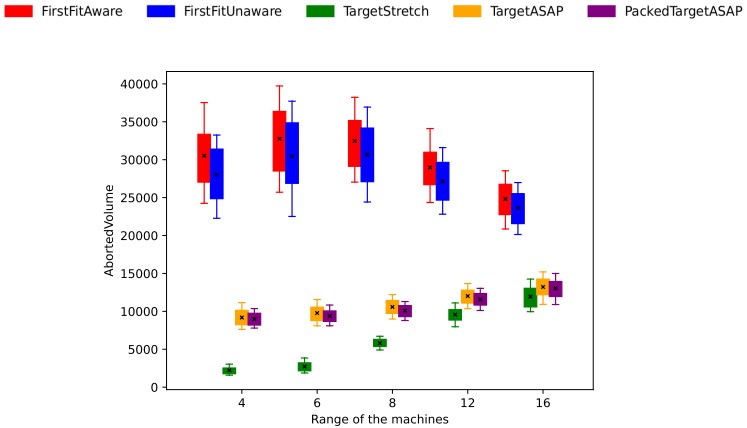
MAXIMUMSTRETCH

Exploring Other Metrics For BORG



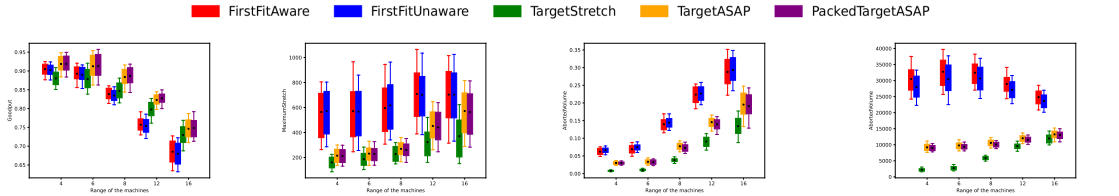
ABORTEDVOLUME

Exploring Other Metrics For BORG



AVERAGEABORTEDTIME

Exploring Other Metrics



GOODPUT MAXIMUMSTRETCH ABORTEDVOLUME AVERAGEABORTEDTIME

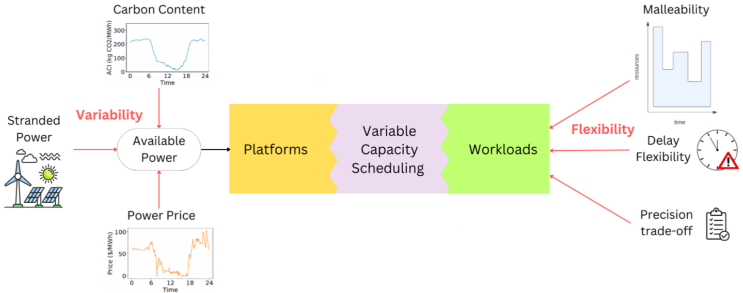
- Increase in range ⇒ Degradation of the metric
- TARGETSTRETCH: lowest maximum stretch, as well as low aborted volume and time
- However, low utilization of machines for TARGETSTRETCH, with low goodput

Conclusion For 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

Back To The Big Picture



Many challenging scheduling problems ☺

Today's case study: restricted instance ☹

Models needed to assess techniques at scale without bias

Research Directions (1/2)

Platforms and resources

- New and more complex definitions of capacity
- Describing resource capacity as a function of time
- Negotiation of capacity between compute management and external factors
- Reporting non-performance attributes

Flexible Workloads

- Flexible start dates, allow migration or deferral
- Flexible resources and accuracy
- Generalize SLAs

Research Directions (2/2)

Scheduling Models and Metrics

- New models for resource variability
- New models for 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
- Preserving fairness among all users
- *Superficial feeling of abundance:* abuse of computational resources, rebound effect

For The Road

Pointers

- Workshop report: *Scheduling Variable Capacity Resources for Sustainability* March 29-31, 2023, U. Chicago Paris Center
- Case-study: *Risk-Aware Scheduling Algorithms for Variable Capacity Resources* PMBS workshop at SC'23