Scientific talk - ROMA team

Variable Capacity Scheduling

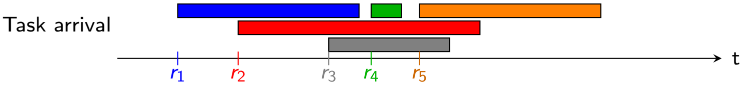
Anne Benoit
LIP, Ecole Normale Supérieure de Lyon
Institut Universitaire de France

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

January 7, 2026 – HCERES

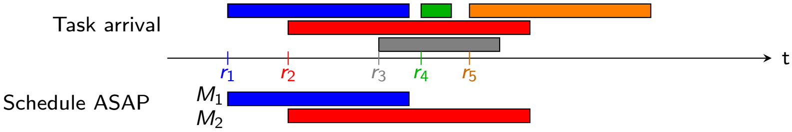
Scheduling

- Online scheduling techniques: decide where and when to execute tasks (jobs) on resources – at the heart of *batch schedulers*
- Basic problem: schedule independent tasks on parallel HPC platforms
- **Objective functions:**
  - Utilization (platform's perspective) – fraction of time when the platform is computing
  - Stretch (user's perspective) – minimize maximum (or average) stretch of tasks, i.e., response time normalized by the task length



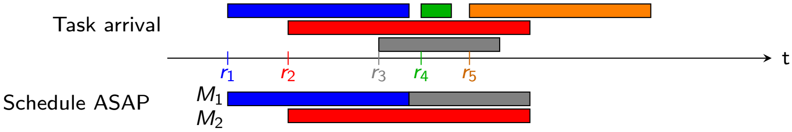
Scheduling

- Online scheduling techniques: decide where and when to execute tasks (jobs) on resources – at the heart of *batch schedulers*
- Basic problem: schedule independent tasks on parallel HPC platforms
- **Objective functions:**
  - Utilization (platform's perspective) – fraction of time when the platform is computing
  - Stretch (user's perspective) – minimize maximum (or average) stretch of tasks, i.e., response time normalized by the task length



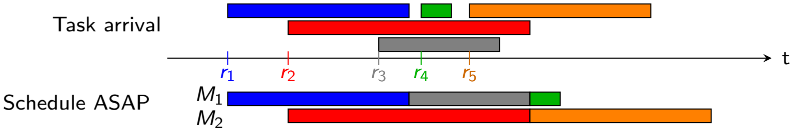
Scheduling

- Online scheduling techniques: decide where and when to execute tasks (jobs) on resources – at the heart of *batch schedulers*
- Basic problem: schedule independent tasks on parallel HPC platforms
- **Objective functions:**
  - Utilization (platform's perspective) – fraction of time when the platform is computing
  - Stretch (user's perspective) – minimize maximum (or average) stretch of tasks, i.e., response time normalized by the task length



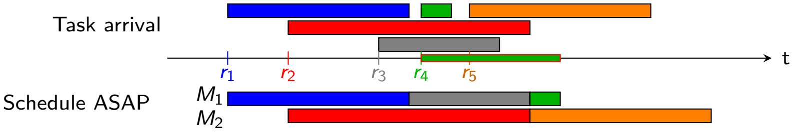
Scheduling

- Online scheduling techniques: decide where and when to execute tasks (jobs) on resources – at the heart of *batch schedulers*
- Basic problem: schedule independent tasks on parallel HPC platforms
- **Objective functions:**
  - Utilization (platform's perspective) – fraction of time when the platform is computing
  - Stretch (user's perspective) – minimize maximum (or average) stretch of tasks, i.e., response time normalized by the task length



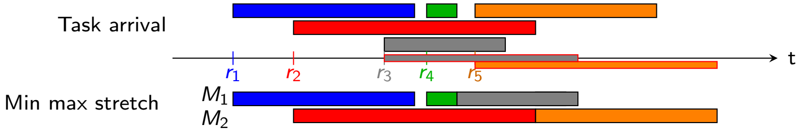
Scheduling

- Online scheduling techniques: decide where and when to execute tasks (jobs) on resources – at the heart of *batch schedulers*
- Basic problem: schedule independent tasks on parallel HPC platforms
- **Objective functions:**
  - Utilization (platform's perspective) – fraction of time when the platform is computing
  - Stretch (user's perspective) – minimize maximum (or average) stretch of tasks, i.e., response time normalized by the task length



Scheduling

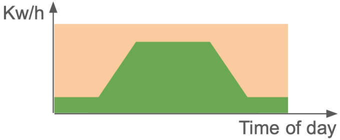
- Online scheduling techniques: decide where and when to execute tasks (jobs) on resources – at the heart of *batch schedulers*
- Basic problem: schedule independent tasks on parallel HPC platforms
- **Objective functions:**
  - Utilization (platform's perspective) – fraction of time when the platform is computing
  - Stretch (user's perspective) – minimize maximum (or average) stretch of tasks, i.e., response time normalized by the task length

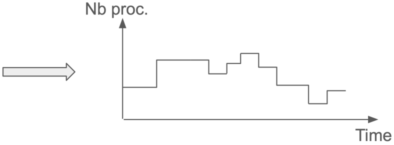


New challenge: Variable capacity

- Today's datacenters assume resource capacity as a fixed quantity:

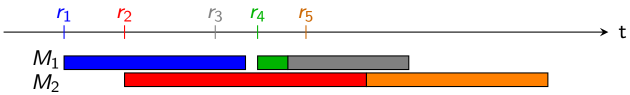
- Emerging approaches ⇒ Variable power
  - Exploit renewable energy
  - Reduce carbon emissions





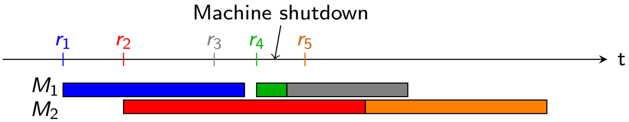
Variable capacity scheduling

- Green computing: the available power evolves in time (solar or wind energy, etc...)
- How to schedule efficiently when the available power varies, which means the number of machines that can be powered varies with time?
- Need to be ready for these variations: if a machine is shut down, need to re-execute its tasks, and start new tasks when a new machine is turned on



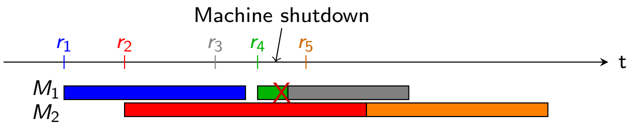
Variable capacity scheduling

- Green computing: the available power evolves in time (solar or wind energy, etc...)
- How to schedule efficiently when the available power varies, which means the number of machines that can be powered varies with time?
- Need to be ready for these variations: if a machine is shut down, need to re-execute its tasks, and start new tasks when a new machine is turned on



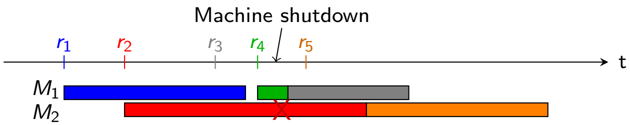
Variable capacity scheduling

- Green computing: the available power evolves in time (solar or wind energy, etc...)
- How to schedule efficiently when the available power varies, which means the number of machines that can be powered varies with time?
- Need to be ready for these variations: if a machine is shut down, need to re-execute its tasks, and start new tasks when a new machine is turned on



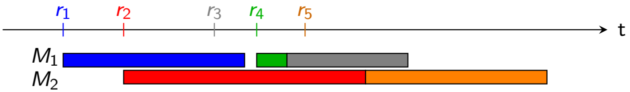
Variable capacity scheduling

- Green computing: the available power evolves in time (solar or wind energy, etc...)
- How to schedule efficiently when the available power varies, which means the number of machines that can be powered varies with time?
- Need to be ready for these variations: if a machine is shut down, need to re-execute its tasks, and start new tasks when a new machine is turned on



Risk aware?

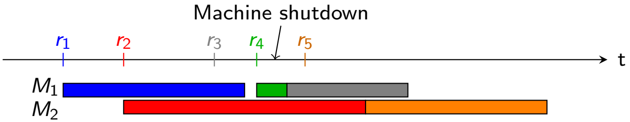
1. **Which machine to shutdown?**



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

Risk aware?

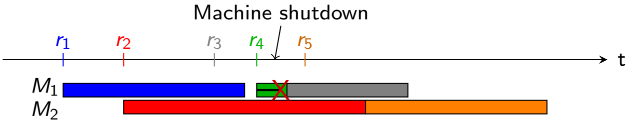
1. **Which machine to shutdown?**



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

Risk aware?

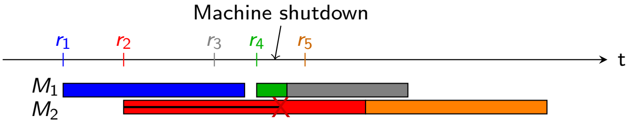
1. **Which machine to shutdown?**



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

Risk aware?

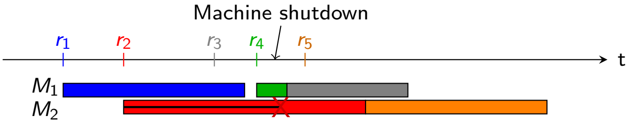
1. **Which machine to shutdown?**



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

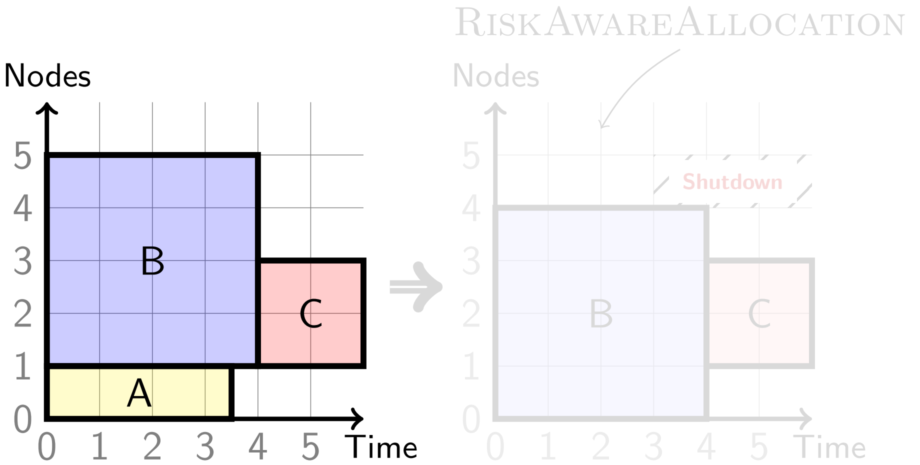
Risk aware?

1. **Which machine to shutdown?**

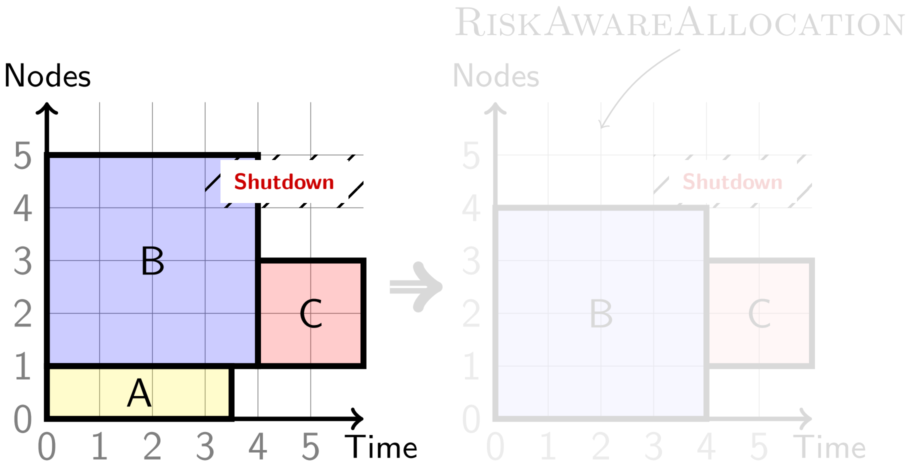


2. **How to schedule jobs to minimize impact?**

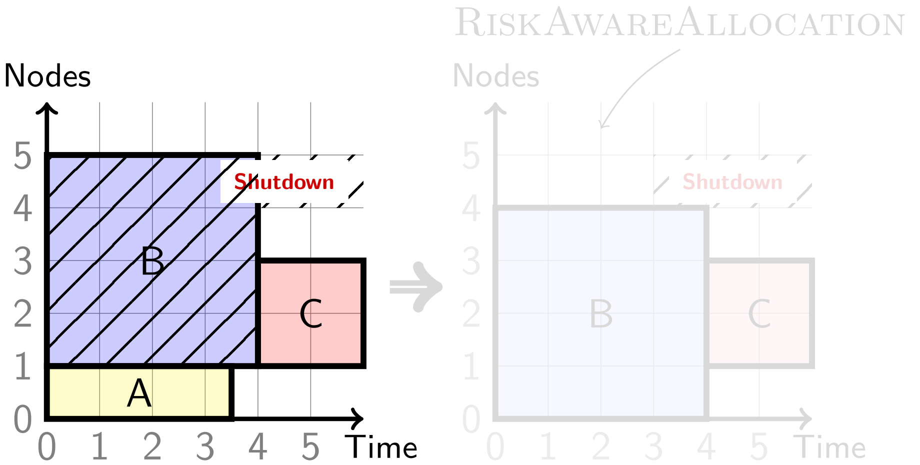
Small example



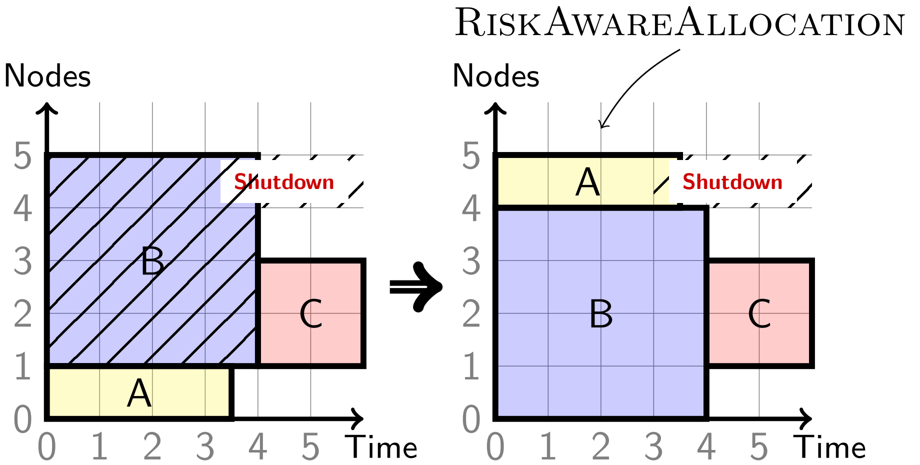
Small example



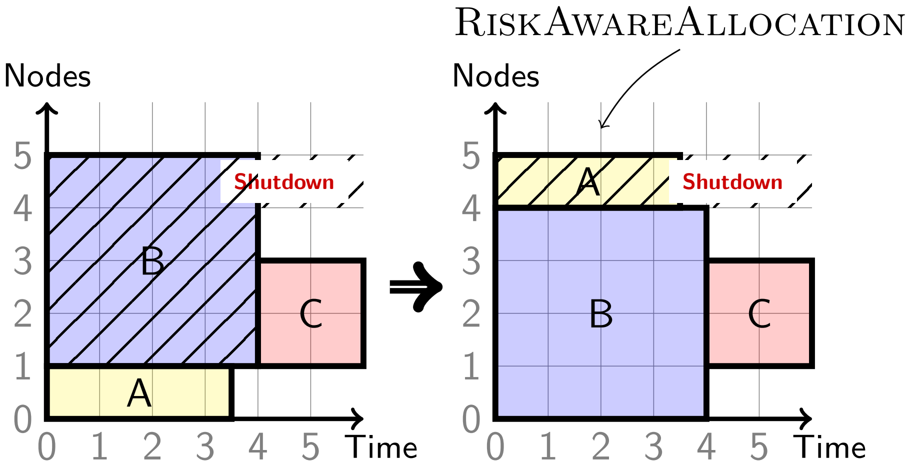
Small example



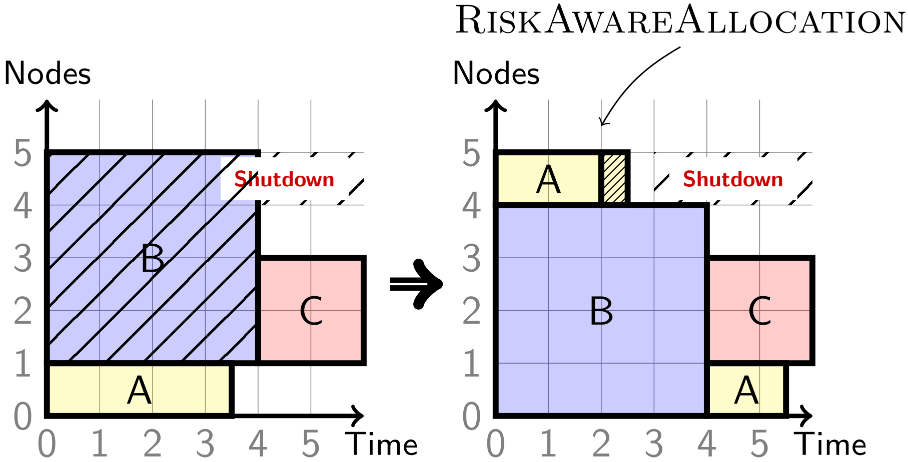
Small example



Small example



Small example



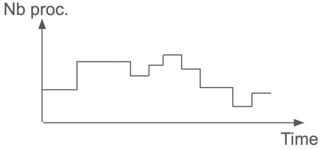
Main questions

- 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?

Outline

Framework

- No possibility to checkpoint jobs or to anticipate a resource variation

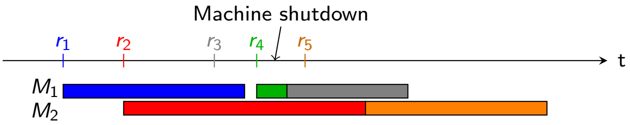


- Set of rigid **jobs**, each using a given number of cores (work w_i on c_i cores)
- Identical multicore **machines**, number of machines alive evolves with time
- Number of alive machines not known until it changes

Design of risk-aware strategies that account for the risk,
assigning new tasks to the *good* target machine,
depending upon the optimization criteria

Objective functions

- Platform utilization: Not a good criteria anymore (some tasks may be interrupted and some work lost)

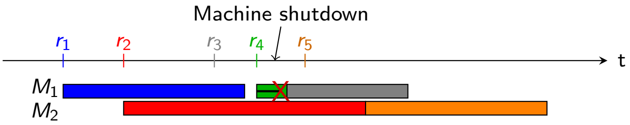


- **Objective function:** Goodput ⇒ *effective* utilization, accounts only for tasks that are completed or still running – fraction of useful work up to time T
  - $\mathcal{J}_{comp,T}$: set of jobs that are completed at time T ($e_i \leq T$)
  - $\mathcal{J}_{started,T}$: set of jobs running and not completed at time T ($s_i \leq T < e_i$)
  - Total number of units of work that can be executed in $[0, T]$: $n_c \sum_{t \in [0,T-1]} M_{alive}(t)$

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

Objective functions

- Platform utilization: Not a good criteria anymore (some tasks may be interrupted and some work lost)

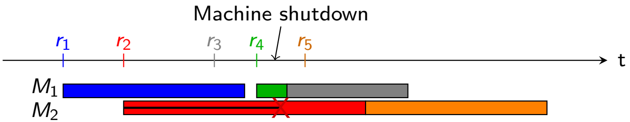


- **Objective function:** Goodput ⇒ *effective* utilization, accounts only for tasks that are completed or still running – fraction of useful work up to time T
  - $\mathcal{J}_{comp,T}$: set of jobs that are completed at time T $(e_i \leq T)$
  - $\mathcal{J}_{started,T}$: set of jobs running and not completed at time T $(s_i \leq T < e_i)$
  - Total number of units of work that can be executed in $[0,T]$: $n_c \sum_{t \in [0,T-1]} M_{alive}(t)$

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

Objective functions

- Platform utilization: Not a good criteria anymore (some tasks may be interrupted and some work lost)

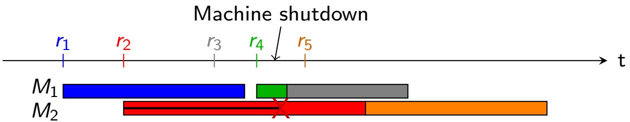


- **Objective function:** Goodput ⇒ *effective* utilization, accounts only for tasks that are completed or still running – fraction of useful work up to time T
  - $\mathcal{J}_{comp,T}$: set of jobs that are completed at time T ($e_i \leq T$)
  - $\mathcal{J}_{started,T}$: set of jobs running and not completed at time T ($s_i \leq T < e_i$)
  - Total number of units of work that can be executed in $[0, T]$: $n_c \sum_{t\in[0,T-1]} M_{alive}(t)$

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

Objective functions

- Platform utilization: Not a good criteria anymore (some tasks may be interrupted and some work lost)



- **Objective function:** Goodput $\Rightarrow$ *effective* utilization, accounts only for tasks that are completed or still running – fraction of useful work up to time T
  -
  -
  -] $M_{alive}(t)$

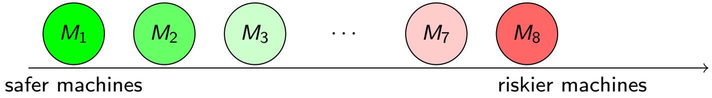
Keep an eye on maximum stretch

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

Algorithms

Main idea:

- Take a decision at each event (task arrival or completion, machine addition or removal)
- Order machines for a guided choice:

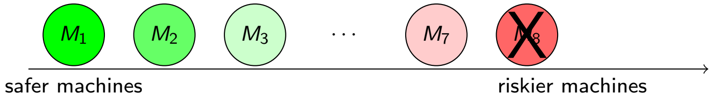


Risk-aware job allocation strategies

Algorithms

Main idea:

- Take a decision at each event (task arrival or completion, machine addition or removal)
- Order machines for a guided choice:



Risk-aware job allocation strategies

Algorithms

Main idea:

- Take a decision at each event (task arrival or completion, machine addition or removal)
- Order machines for a guided choice:

Risk-aware job allocation strategies

Basic heuristics

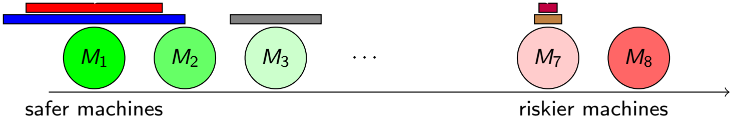
- FirstFitAware (natural approach):
  - Ordered list of machines
  - Jobs mapped to leftmost (safer) machines whenever possible
  - Rightmost (riskier) machines are shutdown whenever necessary
- FirstFitUnaware: Shutdown random machines whenever necessary
- Can we do better than first fit?
  - Interrupting 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

Basic heuristics

- FirstFitAware (natural approach):
  - 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
- Can we do better than first fit?
  - 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

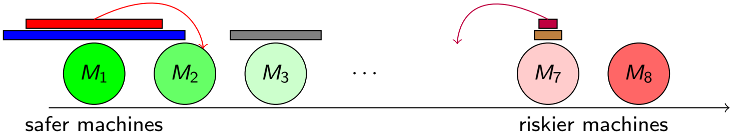
Sophisticated heuristics

- TargetStretch: Add one queue per machine, target value for max stretch
  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



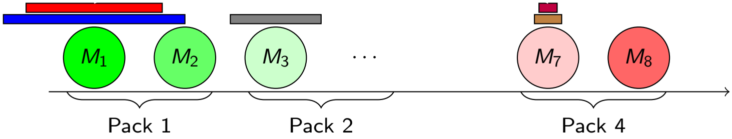
Sophisticated heuristics

- TargetStretch: Add one queue per machine, target value for max stretch
  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



Sophisticated heuristics

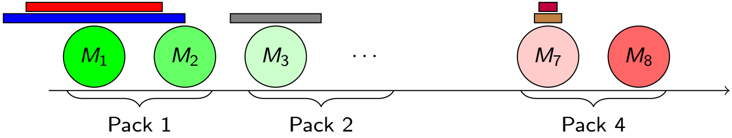
- TargetStretch: Add one queue per machine, target value for max stretch
  potential bad utilization
  No flexibility for mapping to another free machine
- TargetASAP:
  - Start job immediately on target machine or closest machine in neighborhood
  - If not possible, assign on target machine if target 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



Sophisticated heuristics

- TargetStretch: Add one queue per machine, target value for max stretch
  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
  - Othe… …table distance)
- Variant P… …ssign jobs to first mach… …jobs

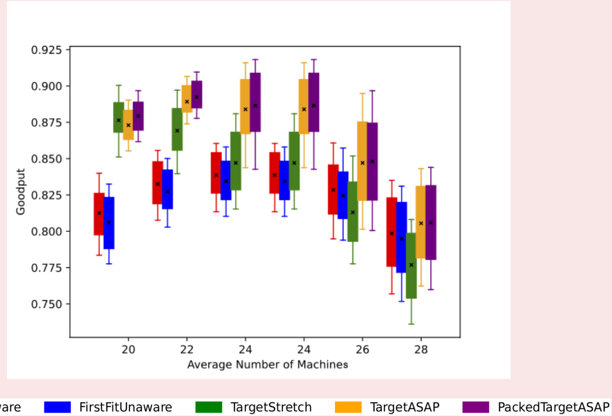
Technical and kind of painful
despite all simplifying hypotheses ☹



Sophisticated heuristics

Simulation results using resource variation trace and job traces (Borg)

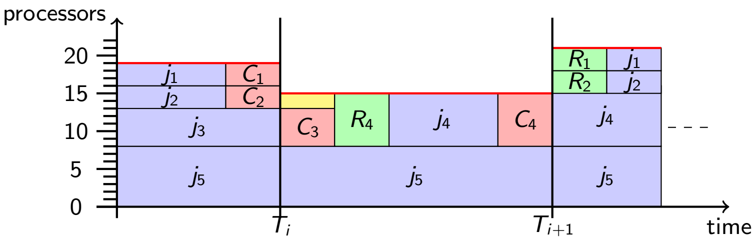
Significant gains over first-fit algorithms: map the right job to the right machine



Outline

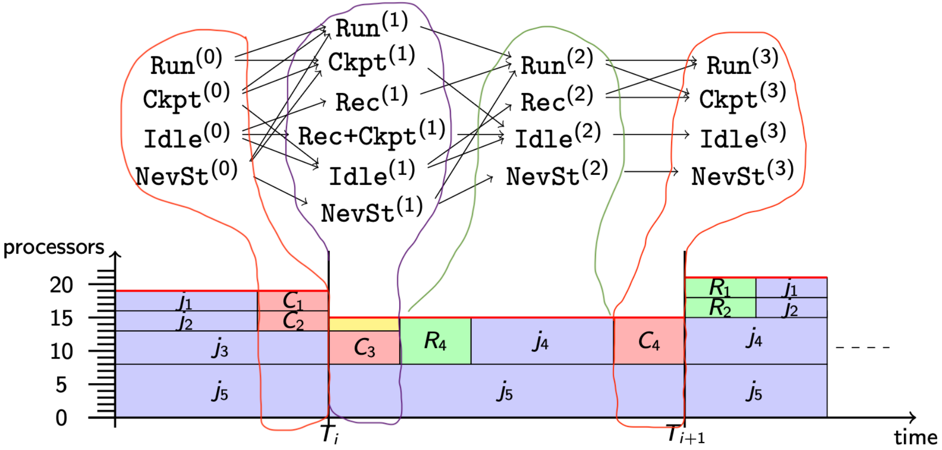
Model

- Avoid losing work: Jobs can be **checkpointed** and recovered
- Maximize goodput – Useful work, excluding time to checkpoint/recover
- **Problem:** Schedule infinite parallel rigid jobs under variable number of processors, during each *section*; maximize goodput and minimum yield (fairness)
- Perfect knowledge of the duration of each section, and bound on #proc difference
- 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

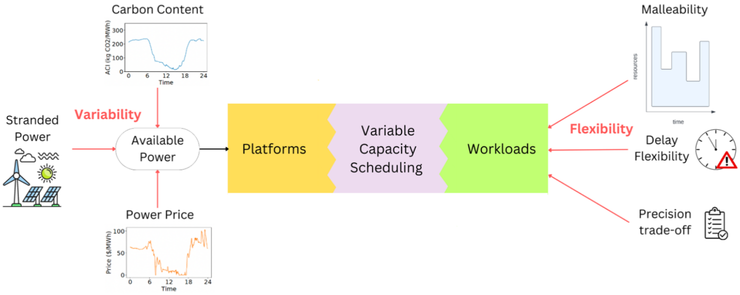


Simulations

- Evaluation on job traces, with both infinite and finite jobs
- Improvement of novel strategies over greedy approaches
- Bi-criteria dynamic programming algorithm DP BiC very efficient

Outline

Conclusion



Many challenging scheduling problems when resources subject to variable capacity ☺

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

Case studies: restricted instances

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

Scheduling Jobs Under a Variable Number of Processors; IEEE Trans. on Parallel and Distributed Systems, 2025

Research directions

Platforms and resources: New and more complex definitions of capacity; understand and model capacity changes

Flexible workloads: Exploit flexible start dates, allow migration or deferral, support multiple precision levels

Scheduling models and metrics: Consider new multi-criteria metrics for both performance and sustainability (including carbon cost); Account for uncertainty

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