How to efficiently use large-scale systems? Dealing with errors and variable capacity

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

Joint work with Y. Robert, L. Perotin, J. Cendrier, F. Vivien, A. Tremodeux, A. Cavelan (ENS Lyon), H. Sun (U. Kansas), T. Herault (Inria), A. A. Chien, R. Wijayawardana, C. Zhang (U. Chicago)

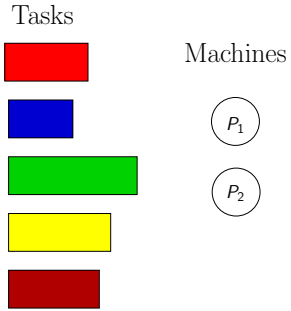
GrAPL Workshop on Graphs, Architectures, Programming, and Learning
Colocated with IPDPS, New Orleans, May 26, 2026

Motivation

Scheduling: Allocate resources to applications to optimize some performance metrics

- Resources: Large-scale distributed systems with millions of components
- Applications: Parallel applications, expressed as a set of tasks, or divisible application with some work to complete
- Performance metrics: Of course we are concerned with the performance of the applications, but also with resilience and energy consumption

Classical scheduling problems



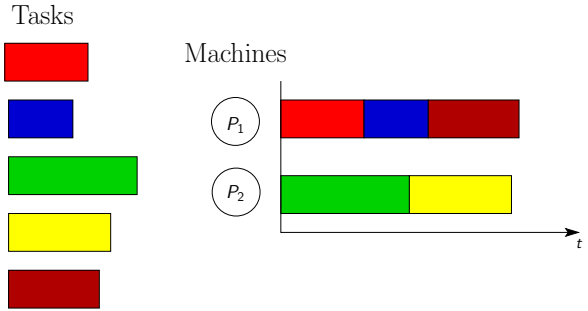
Objectives:

- Minimizing total execution time (C_{max})
- Minimizing weighted sum of execution times $\sum_i w_i C_i$

Results:

- NP-completeness or optimal polynomial-time algorithms
- Approximation algorithms, (in-)approximation bounds

Classical scheduling problems



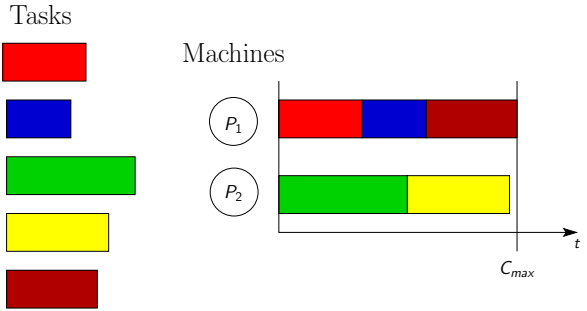
Objectives:

- Minimizing total execution time (C_{max})
- Minimizing weighted sum of execution times $\sum_i w_i C_i$

Results:

- NP-completeness or optimal polynomial-time algorithms
- Approximation algorithms, (in-)approximation bounds

Classical scheduling problems



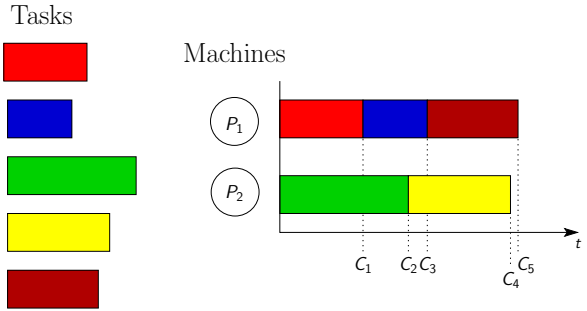
Objectives:

- Minimizing total execution time (C_{max})
- Minimizing weighted sum of execution times $\sum_i w_i C_i$

Results:

- NP-completeness or optimal polynomial-time algorithms
- Approximation algorithms, (in-)approximation bounds

Classical scheduling problems



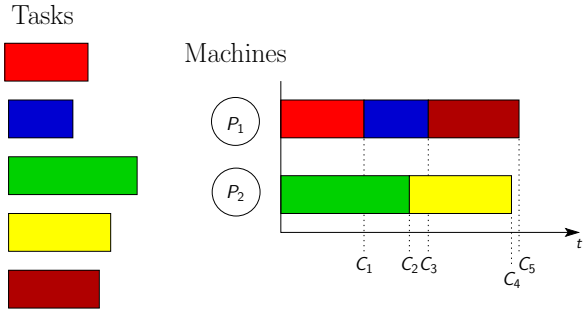
Objectives:

- Minimizing total execution time (C_{max})
- Minimizing weighted sum of execution times $\sum_i w_i C_i$

Results:

- NP-completeness or optimal polynomial-time algorithms
- Approximation algorithms, (in-)approximation bounds

Classical scheduling problems



Objectives:

- Minimizing total execution time (C_{max})
- Minimizing weighted sum of execution times $\sum_i w_i C_i$

Results:

- NP-completeness or optimal polynomial-time algorithms
- Approximation algorithms, (in-)approximation bounds

Dealing with failures

- Consider one processor (e.g. in your laptop)
  - Mean Time Between Failures (MTBF) = 100 years
  - (Almost) no failures in practice ☺

  Why bother about failures?

- **Theorem:** The MTBF decreases linearly with the number of processors! With 36500 processors:
  - MTBF = 1 day
  - A failure every day on average!

A large simulation can run for weeks, hence it will face failures ☹

Dealing with failures

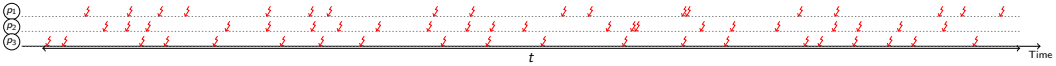
- Consider one processor (e.g. in your laptop)
  - Mean Time Between Failures (MTBF) = 100 years
  - (Almost) no failures in practice ☺

  Why bother about failures?

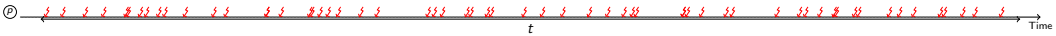
- **Theorem:** The MTBF decreases linearly with the number of processors! With 36500 processors:
  - MTBF = 1 day
  - A failure every day on average!

A large simulation can run for weeks, hence it will face failures ☹

Intuition



If three processors have around 20 faults during a time t ($\mu = \frac{t}{20}$)...

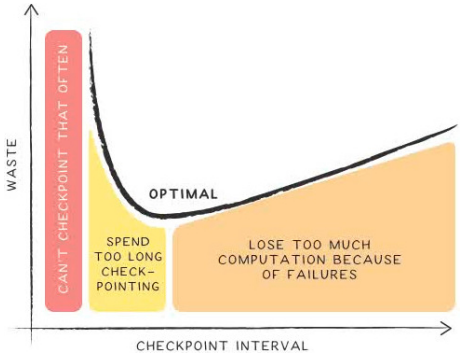


...during the same time, the platform has around 60 faults ($\mu_p = \frac{t}{60}$)

So, how to deal with failures?

Failures usually handled by adding redundancy:

- Replicate the work (for instance, use only half of the processors, and the other half is used to redo the same computation)
- Checkpoint the application: Periodically save the state of the application on stable storage, so that we can restart in case of failure without loosing everything



Another crucial issue: Energy consumption

"The internet begins with coal"

- Nowadays: more than 90 billion kilowatt-hours of electricity a year; requires 34 giant (500 megawatt) coal-powered plants, and produces huge CO_2 emissions
- Explosion of artificial intelligence; AI is hungry for processing power! Need to double data centers in next four years
  $\rightarrow$ how to get enough power?
- Failures: Redundant work consumes even more energy

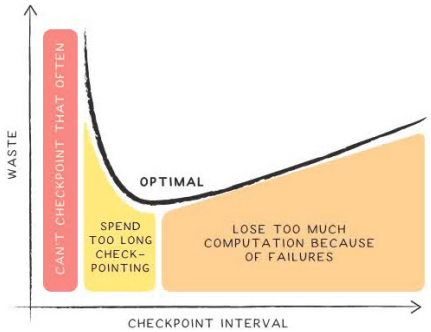
Energy and power awareness $\rightsquigarrow$ crucial for both environmental and economical reasons

Outline

Introduction to resilience

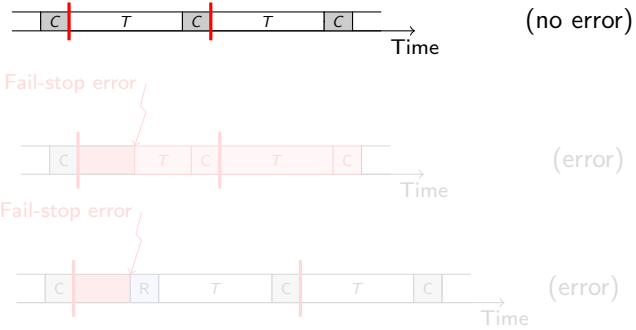
- Fail-stop errors:
  - Component failures (node, network, power, ...)
  - Application fails and data is lost
- Silent data corruptions:
  - Bit flip (Disk, RAM, Cache, Bus, ...)
  - Detection is not immediate, and we may get wrong results

How often should we checkpoint to minimize the waste, i.e., the time lost because of resilience techniques and failures?



Coping with fail-stop errors

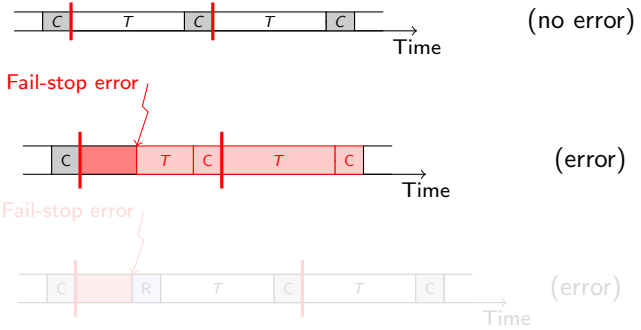
Periodic checkpoint, rollback, and recovery:



- Coordinated checkpointing (the platform is a giant macro-processor)
- Assume instantaneous interruption and detection
- Rollback to last checkpoint and re-execute

Coping with fail-stop errors

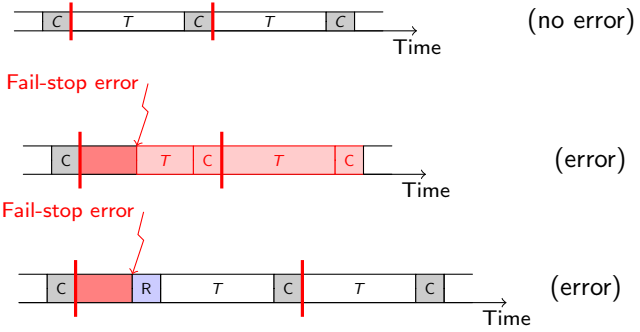
Periodic checkpoint, rollback, and recovery:



- Coordinated checkpointing (the platform is a giant macro-processor)
- Assume instantaneous interruption and detection
- Rollback to last checkpoint and re-execute

Coping with fail-stop errors

Periodic checkpoint, rollback, and recovery:



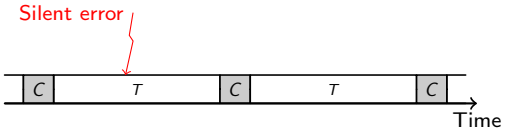
- Coordinated checkpointing (the platform is a giant macro-processor)
- Assume instantaneous interruption and detection
- Rollback to last checkpoint and re-execute

Coping with silent errors

Silent error = detection latency

Error is detected only when corrupted data is activated

Same approach?



Keep multiple checkpoints?

Which checkpoint to recover from?

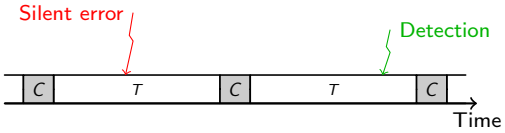
Need an active method to detect silent errors!

Coping with silent errors

Silent error = detection latency

Error is detected only when corrupted data is activated

Same approach?



Keep multiple checkpoints?

Which checkpoint to recover from?

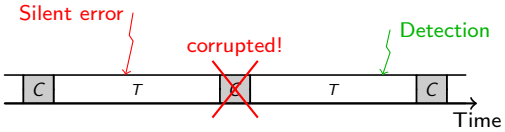
Need an active method to detect silent errors!

Coping with silent errors

Silent error = detection latency

Error is detected only when corrupted data is activated

Same approach?



Keep multiple checkpoints?

Which checkpoint to recover from?

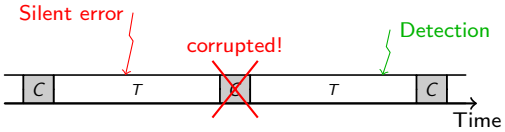
Need an active method to detect silent errors!

Coping with silent errors

Silent error = detection latency

Error is detected only when corrupted data is activated

Same approach?



Keep multiple checkpoints?

Which checkpoint to recover from?

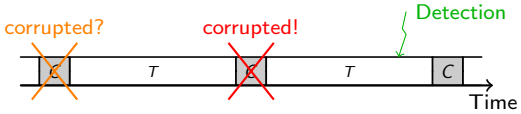
Need an active method to detect silent errors!

Coping with silent errors

Silent error = detection latency

Error is detected only when corrupted data is activated

Same approach?



Keep multiple checkpoints?

Which checkpoint to recover from?

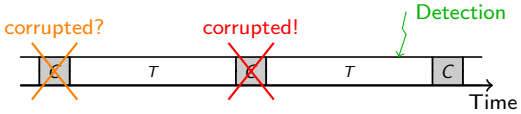
Need an active method to detect silent errors!

Coping with silent errors

Silent error = detection latency

Error is detected only when corrupted data is activated

Same approach?



Keep multiple checkpoints?

Which checkpoint to recover from?

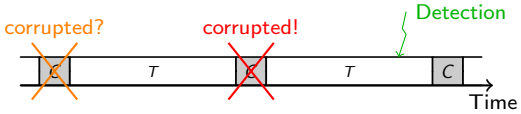
Need an active method to detect silent errors!

Coping with silent errors

Silent error = detection latency

Error is detected only when corrupted data is activated

Same approach?



Keep multiple checkpoints?

Which checkpoint to recover from?

Need an active method to detect silent errors!

Methods for detecting silent errors

General-purpose approaches

- Replication [*Fiala et al. 2012*] or triple modular redundancy and voting [*Lyons and Vanderkulk 1962*]

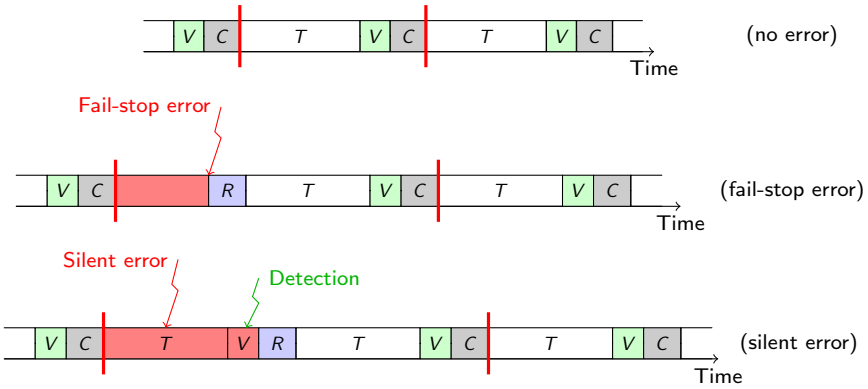
Application-specific approaches

- Algorithm-based fault tolerance (ABFT): checksums in dense matrices Limited to one error detection and/or correction in practice [*Huang and Abraham 1984*]
- Partial differential equations (PDE): use lower-order scheme as verification mechanism [*Benson, Schmit and Schreiber 2014*]
- Generalized minimal residual method (GMRES): inner-outer iterations [*Hoemmen and Heroux 2011*]
- Preconditioned conjugate gradients (PCG): orthogonalization check every k iterations, re-orthogonalization if problem detected [*Sao and Vuduc 2013, Chen 2013*]

Data-analytics approaches

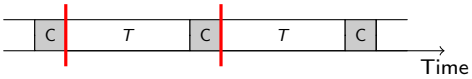
- Dynamic monitoring of HPC datasets based on physical laws (e.g., temperature limit, speed limit) and space or temporal proximity [*Bautista-Gomez and Cappello 2014*]
- Time-series prediction, spatial multivariate interpolation [*Di et al. 2014*]

Coping with fail-stop and silent errors



What is the optimal checkpointing period?

Optimization objective (1/2)



- T is the **pattern length** (time without failures)
- C is the checkpoint cost
- $\mathbb{E}(T)$ is the expected execution time of the pattern

By definition, the overhead of the pattern is defined as:

$$\mathbb{H}(T) = \frac{\mathbb{E}(T)}{T} - 1$$

The overhead measures the fraction of **extra time** due to:

- Checkpoints
- Recoveries and re-executions (failures)

The goal is to minimize the quantity: $\mathbb{H}(T)$

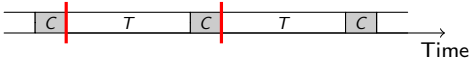
Optimization objective (2/2)

- Goal: Find the **optimal pattern length** T^*, so that the overhead is minimized
- Overhead: $\mathbb{H}(T) = \frac{\mathbb{E}(T)}{T} - 1$

1. Compute expected execution time $\mathbb{E}(T)$ (exact formula)
2. Compute overhead $\mathbb{H}(T)$ (first-order approximation)
3. Derive optimal T^*: fail-stop errors
4. Derive optimal T^*: silent errors
5. Derive optimal T^*: both

1. Expected execution time $\mathbb{E}(T)$

- T: Pattern length
- C: Checkpoint time
- R: Recovery time
- $\lambda^f = \frac{1}{\mu^f}$: Fail-stop error rate



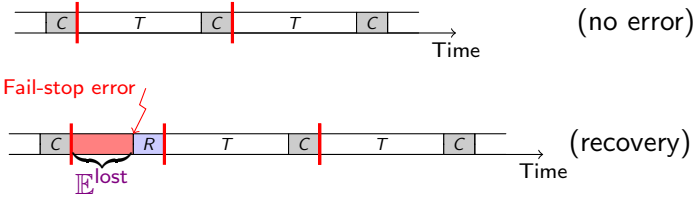
(no error)

$$\mathbb{E}(T) = \mathbb{P}_{no-error}\,(T + C)$$

$$+$$

1. Expected execution time $\mathbb{E}(T)$

- T: Pattern length
- C: Checkpoint time
- R: Recovery time
- $\lambda^f = \frac{1}{\mu^f}$: Fail-stop error rate



$$\mathbb{E}(T) = \mathbb{P}_{no-error}\,(T + C) + \mathbb{P}_{error}\left(\mathbb{E}^{\text{lost}} + R + \mathbb{E}(T)\right)$$

1. Expected execution time $\mathbb{E}(T)$

Assume that failures follow an **exponential distribution** $\text{Exp}(\lambda^f)$

- Independent errors (memoryless property)

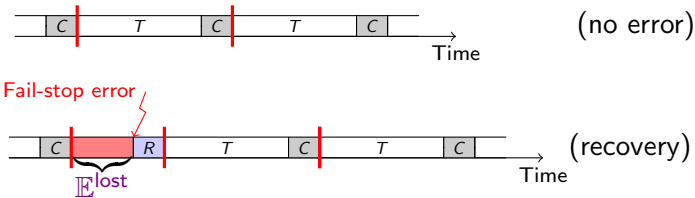
There is at least one error before time t with probability:

$$\mathbb{P}(X \leq t) = 1 - e^{-\lambda^f t} \quad \text{(cdf)}$$

Probability of failure / no-failure

- $\mathbb{P}_{error} = 1 - e^{-\lambda^f T}$
- $\mathbb{P}_{no-error} = e^{-\lambda^f T}$

1. Expected execution time $\mathbb{E}(T)$



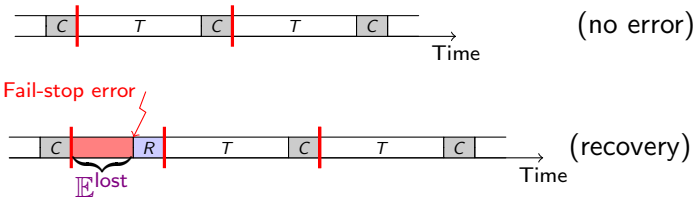
$$\mathbb{E}(T) = e^{-\lambda^f T}(T + C) + (1 - e^{-\lambda^f T})\left(\mathbb{E}^{\text{lost}} + R + \mathbb{E}(T)\right)$$
$$= T + C + (e^{\lambda^f T} - 1)\left(\mathbb{E}^{\text{lost}} + R\right)$$

$\mathbb{E}^{\text{lost}}$ is the time lost when the failure strikes:

$$\mathbb{E}^{\text{lost}} = \int_0^\infty t\mathbb{P}(X = t | X < T)dt = \frac{1}{\lambda^f} - \frac{T}{e^{\lambda^f T} - 1} = \frac{T}{2} + o(\lambda^f T)$$

- **We lose half the pattern upon failure (in expectation)!**

1. Expected execution time $\mathbb{E}(T)$



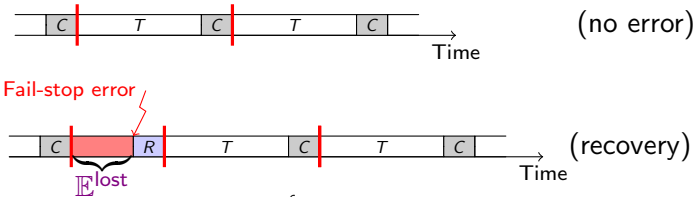
$$\mathbb{E}(T) = e^{-\lambda^f T}(T + C) + (1 - e^{-\lambda^f T})\left(\mathbb{E}^{\text{lost}} + R + \mathbb{E}(T)\right)$$
$$= T + C + (e^{\lambda^f T} - 1)\left(\mathbb{E}^{\text{lost}} + R\right)$$

$\mathbb{E}^{\text{lost}}$ is the time lost when the failure strikes:

$$\mathbb{E}^{\text{lost}} = \int_0^\infty t\mathbb{P}(X = t | X < T)dt = \frac{1}{\lambda^f} - \frac{T}{e^{\lambda^f T} - 1} = \frac{T}{2} + o(\lambda^f T)$$

- **We lose half the pattern upon failure (in expectation)!**

2. Compute overhead $\mathbb{H}(T)$



We use Taylor series to approximate $e^{-\lambda^f T}$ up to first-order terms:

$$e^{-\lambda^f T} = 1 - \lambda^f T + o(\lambda^f T)$$

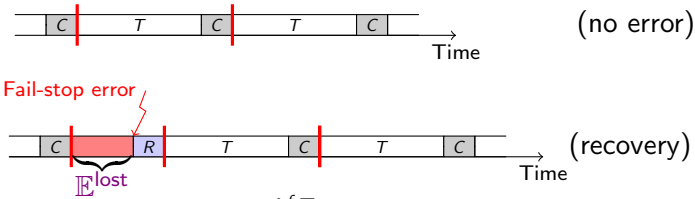
Works well provided that $\lambda^f << T, C, R$

$$\mathbb{E}(T) = T + C + \lambda^f T \left(\frac{T}{2} + R \right) + o(\lambda^f T)$$

Finally, we get the overhead of the pattern:

$$\mathbb{H}(T) = \frac{C}{T} + \lambda^f \frac{T}{2} + o(\lambda^f T)$$

2. Compute overhead $\mathbb{H}(T)$



We use Taylor series to approximate $e^{-\lambda^f T}$ up to first-order terms:

$$e^{-\lambda^f T} = 1 - \lambda^f T + o(\lambda^f T)$$

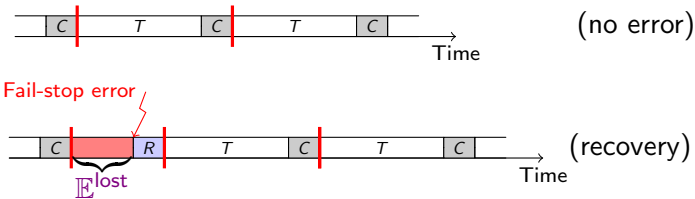
Works well provided that $\lambda^f << T, C, R$

$$\mathbb{E}(T) = T + C + \lambda^f T \left(\frac{T}{2} + R \right) + o(\lambda^f T)$$

Finally, we get the overhead of the pattern:

$$\mathbb{H}(T) = \frac{C}{T} + \lambda^f \frac{T}{2} + o(\lambda^f T)$$

3. Derive optimal T^*: Fail-stop errors



$$\mathbb{H}(T) = \frac{C}{T} + \lambda^f \frac{T}{2} + o(\lambda^f T)$$

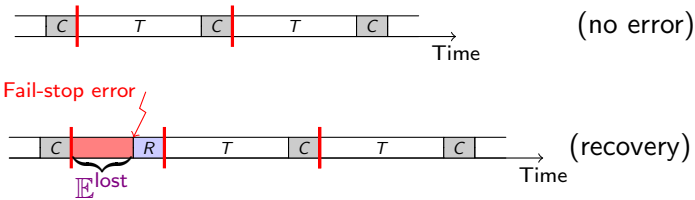
We solve:

$$\frac{\partial \mathbb{H}(T)}{\partial T} = -\frac{C}{T^2} + \frac{\lambda^f}{2} = 0$$

Finally, we retrieve:

$$T^* = \sqrt{\frac{2C}{\lambda^f}} = \sqrt{2\mu^f C}$$

3. Derive optimal T^*: Fail-stop errors



$$\mathbb{H}(T) = \frac{C}{T} + \lambda^f \frac{T}{2} + o(\lambda^f T)$$

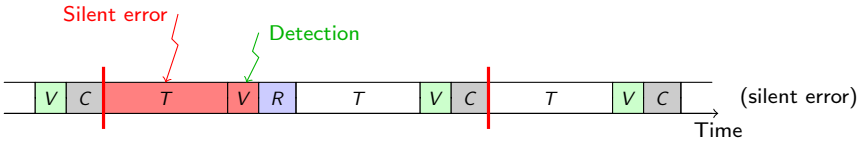
We solve:

$$\frac{\partial \mathbb{H}(T)}{\partial T} = -\frac{C}{T^2} + \frac{\lambda^f}{2} = 0$$

Finally, we retrieve:

$$T^* = \sqrt{\frac{2C}{\lambda^f}} = \sqrt{2\mu^f C}$$

4. Derive optimal T^*: Silent errors



Similar to fail-stop except:

- $\lambda^f \rightarrow \lambda^s$
- $\mathbb{E}^{\text{lost}} = T$
- V: verification time

Using the same approach:

$$\mathbb{H}(T) = \frac{C+V}{T} + \underbrace{\lambda^s T}_{\textit{silent}} + o(\lambda^s T)$$

5. Derive optimal T^*: Both errors

$$\mathbb{H}(T) = \frac{C+V}{T} + \underbrace{\lambda^f \frac{T}{2}}_{fail-stop} + \underbrace{\lambda^s T}_{silent} + o(\lambda T)$$

First-order approximations [Young 1974, Daly 2006, AB et al. 2016]

	Fail-stop errors	Silent errors	Both errors
Pattern	$T + C$	$T + V + C$	$T + V + C$
Optimal T^*	$\sqrt{\frac{C}{\frac{\lambda^f}{2}}}$	$\sqrt{\frac{V+C}{\lambda^s}}$	$\sqrt{\frac{V+C}{\lambda^s + \frac{\lambda^f}{2}}}$
Overhead $\mathbb{H}^*$	$2\sqrt{\frac{\lambda^f}{2} C}$	$2\sqrt{\lambda^s (V+C)}$	$2\sqrt{\left(\lambda^s + \frac{\lambda^f}{2}\right)(V+C)}$

5. Derive optimal T^*: Both errors

$$\mathbb{H}(T) = \frac{C+V}{T} + \underbrace{\lambda^f \frac{T}{2}}_{fail-stop} + \underbrace{\lambda^s T}_{silent} + o(\lambda T)$$

First-order approximations [Young 1974, Daly 2006, AB et al. 2016]

	Fail-stop errors	Silent errors	Both errors
Pattern	$T + C$	$T + V + C$	$T + V + C$
Optimal T^*	$\sqrt{\frac{C}{\frac{\lambda^f}{2}}}$	$\sqrt{\frac{V+C}{\lambda^s}}$	$\sqrt{\frac{V+C}{\lambda^s+\frac{\lambda^f}{2}}}$
Overhead $\mathbb{H}^*$	$2\sqrt{\frac{\lambda^f}{2}C}$	$2\sqrt{\lambda^s(V+C)}$	$2\sqrt{\left(\lambda^s+\frac{\lambda^f}{2}\right)(V+C)}$

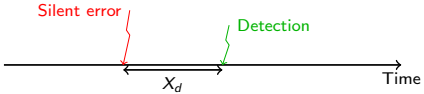
Idealist model?

- To deal with silent errors, do we have/need perfect detectors?
- Replication: Execute for a time T, and re-execute until getting the same result twice, then validate result and checkpoint

Two different errors never lead to the same (incorrect) result

- Use of partial detectors, with recall < 1, i.e., false negatives – some errors may not be immediately detected

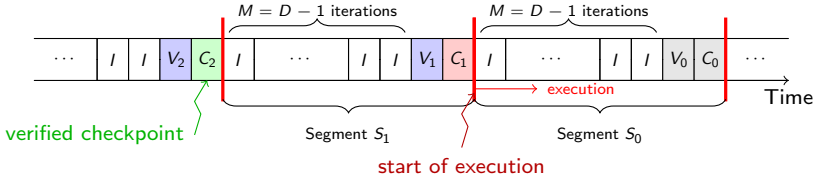
Errors eventually detected, say after D iterations



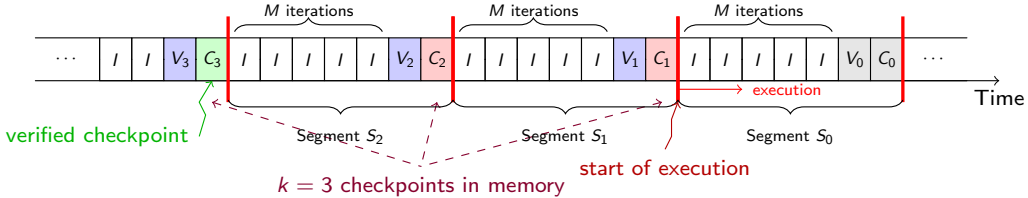
Detection distance: probability distribution X_d with bounded support $[1, D]$

Partial detectors

- Simple scheme

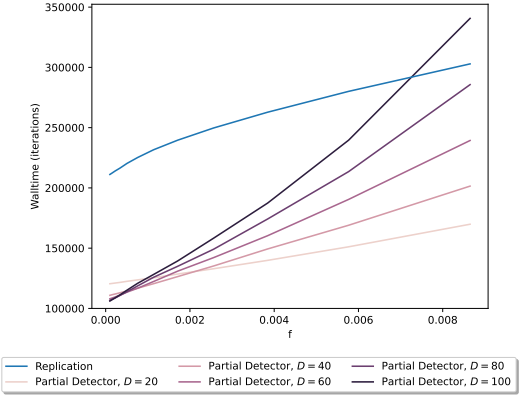


- General scheme



Analysis and results

- After painful computations, we get the expected time to successfully process segment S_0, take a new checkpoint, and verify the next checkpoint
- Compare with replication? Recall 0.4 (captures only 40% of errors), increasing error risk f (probability of error during one iteration)



Summary and need for trade-offs

- Two major challenges for Exascale systems:
  - Resilience: need to handle failures
  - Energy: need to reduce energy consumption
- The main objective is often performance, such as execution time, but other criteria must be accounted for
- Many models for which we have the answer:
  - Optimal checkpointing period, with fail-stop / silent errors
  - Optimal checkpointing period to minimize energy consumption
  - Use of replication or non-perfect detectors
- Still a lot of challenges to address, and techniques to be developed for many kinds of high-performance applications, making trade-offs between performance, reliability, and energy consumption
- Emerging challenge: Variable capacity scheduling

Summary and need for trade-offs

- Two major challenges for Exascale systems:
  - Resilience: need to handle failures
  - Energy: need to reduce energy consumption
- The main objective is often performance, such as execution time, but other criteria must be accounted for
- Many models for which we have the answer:
  - Optimal checkpointing period, with fail-stop / silent errors
  - Optimal checkpointing period to minimize energy consumption
  - Use of replication or non-perfect detectors
- Still a lot of challenges to address, and techniques to be developed for many kinds of high-performance applications, making trade-offs between performance, reliability, and energy consumption
- Emerging challenge: Variable capacity scheduling

Summary and need for trade-offs

- Two major challenges for Exascale systems:
  - Resilience: need to handle failures
  - Energy: need to reduce energy consumption
- The main objective is often performance, such as execution time, but other criteria must be accounted for
- Many models for which we have the answer:
  - Optimal checkpointing period, with fail-stop / silent errors
  - Optimal checkpointing period to minimize energy consumption
  - Use of replication or non-perfect detectors
- Still a lot of challenges to address, and techniques to be developed for many kinds of high-performance applications, making trade-offs between performance, reliability, and energy consumption
- Emerging challenge: Variable capacity scheduling

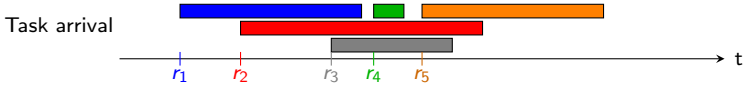
Summary and need for trade-offs

- Two major challenges for Exascale systems:
  - Resilience: need to handle failures
  - Energy: need to reduce energy consumption
- The main objective is often performance, such as execution time, but other criteria must be accounted for
- Many models for which we have the answer:
  - Optimal checkpointing period, with fail-stop / silent errors
  - Optimal checkpointing period to minimize energy consumption
  - Use of replication or non-perfect detectors
- Still a lot of challenges to address, and techniques to be developed for many kinds of high-performance applications, making trade-offs between performance, reliability, and energy consumption
- Emerging challenge: Variable capacity scheduling

Outline

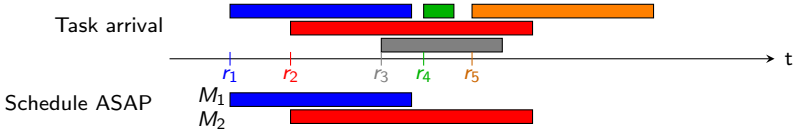
Scheduling

- 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



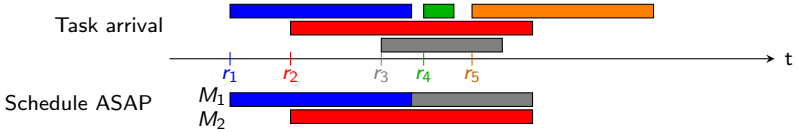
Scheduling

- 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



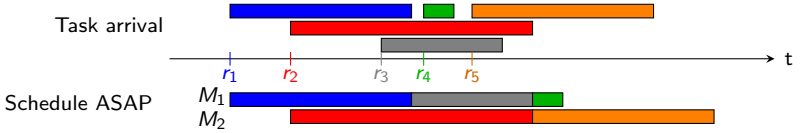
Scheduling

- 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



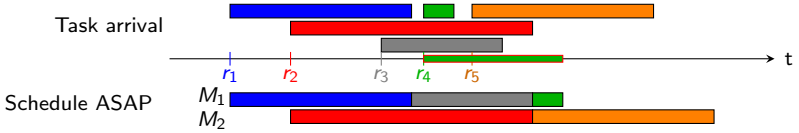
Scheduling

- 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



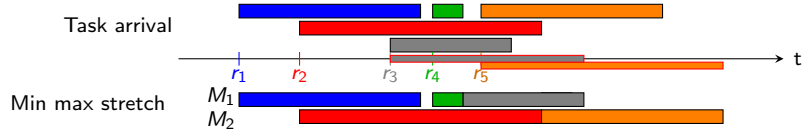
Scheduling

- 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

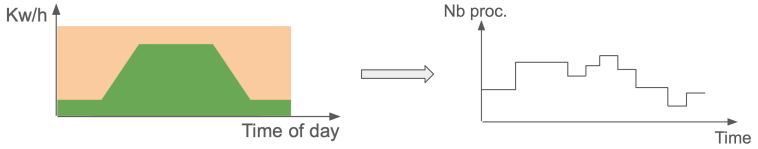
- Online scheduling techniques: decide where and when to execute tasks (jobs) on resources – at the heart of *batch schedulers*
- Basic problem: schedule independent tasks on parallel HPC platforms
- **Objective functions:**
  - Utilization (platform's perspective) – fraction of time when the platform 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

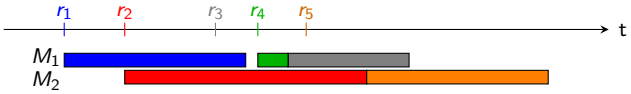
- Today's datacenters assume resource capacity as a fixed quantity:

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



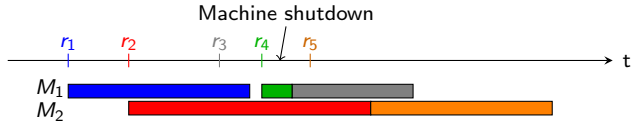
Variable capacity scheduling

- 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



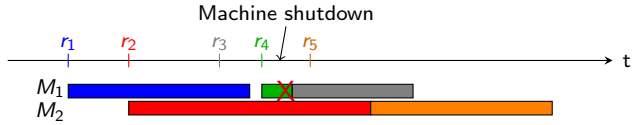
Variable capacity scheduling

- 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



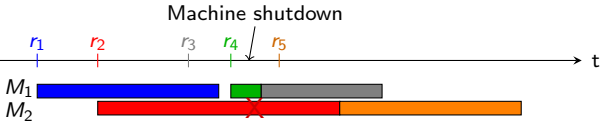
Variable capacity scheduling

- 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



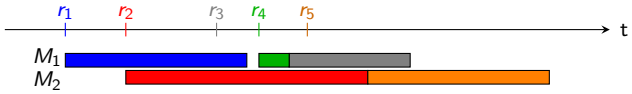
Variable capacity scheduling

- 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?

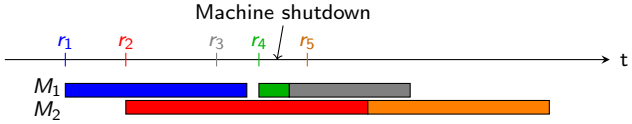
1. **Which machine to shutdown?**



2. How to schedule jobs to minimize impact?

Risk aware?

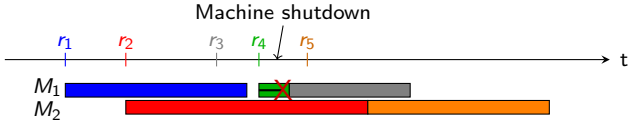
1. **Which machine to shutdown?**



2. How to schedule jobs to minimize impact?

Risk aware?

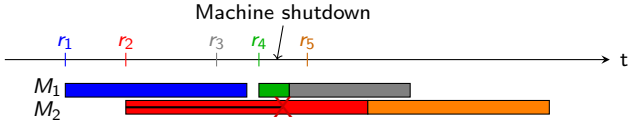
1. **Which machine to shutdown?**



2. How to schedule jobs to minimize impact?

Risk aware?

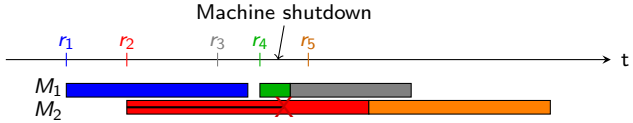
1. **Which machine to shutdown?**



2. How to schedule jobs to minimize impact?

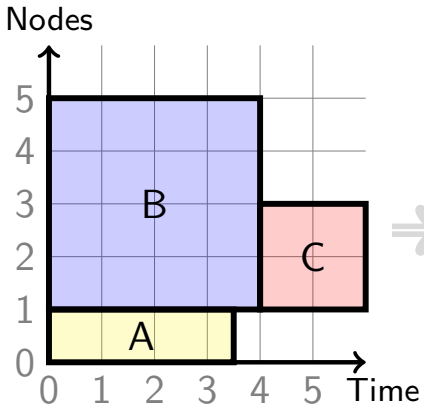
Risk aware?

1. **Which machine to shutdown?**

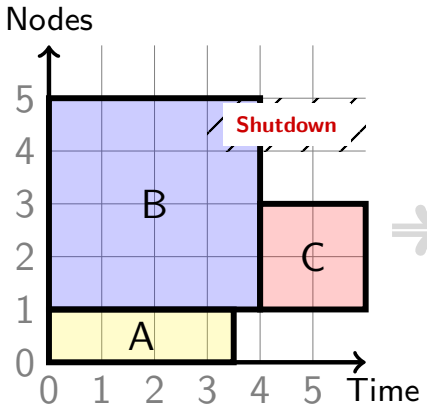


2. **How to schedule jobs to minimize impact?**

Small example

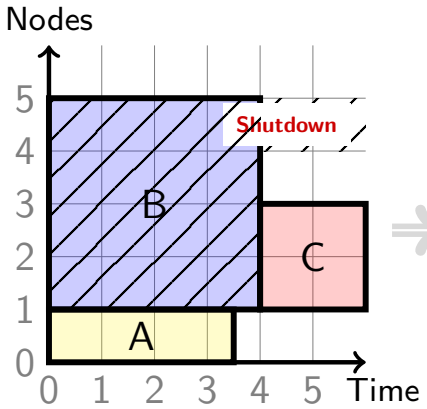


Small example



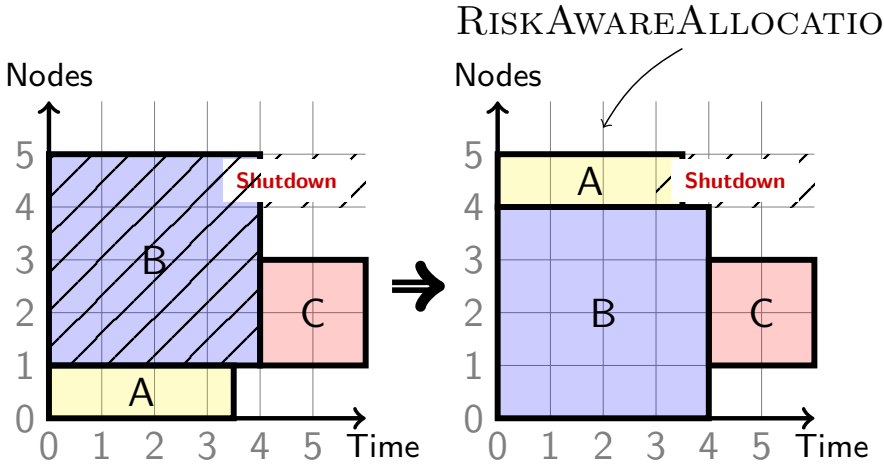
RiskAwareAllocation

Small example

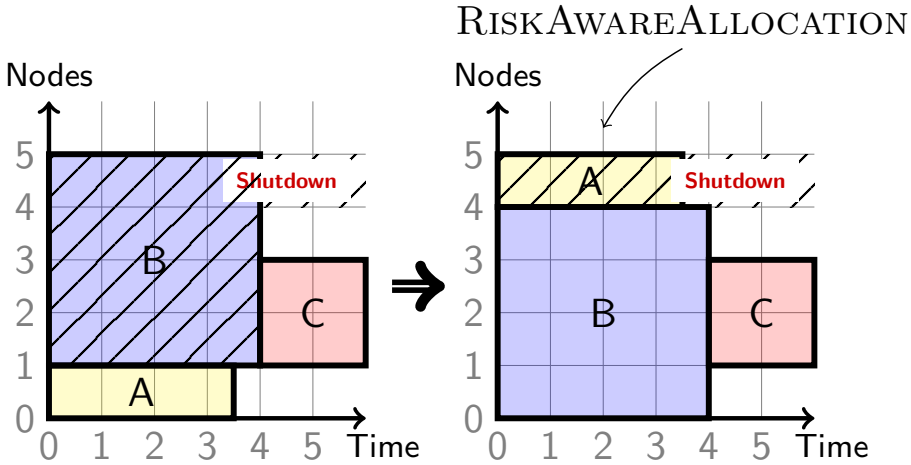


RiskAwareAllocation

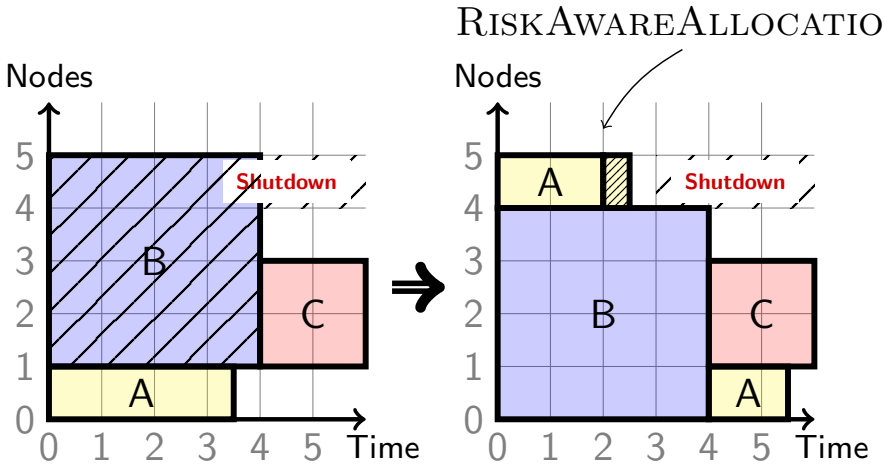
Small example



Small example



Small example



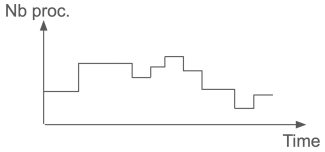
Main questions

- 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

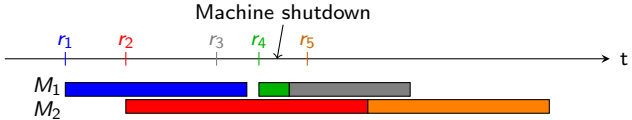


- Set of rigid **jobs**, each using a given number of cores (work w_i on c_i cores)
- Identical multicore **machines**, number of machines alive evolves with time
- Number of alive machines not known until it changes

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)

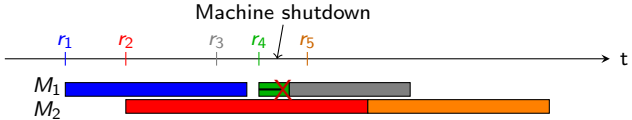


- **Objective function:** Goodput $\Rightarrow$ *effective* utilization, accounts only for tasks that are completed or still running – fraction of useful work up to time T
  - $\mathcal{J}_{comp,T}$: set of jobs that are completed at time T ($e_i \leq T$)
  - $\mathcal{J}_{started,T}$: set of jobs running 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)

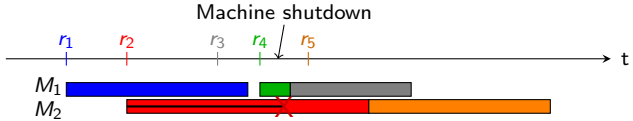


- **Objective function:** Goodput $\Rightarrow$ *effective* utilization, accounts only for tasks that are completed or still running – fraction of useful work up to time T
  - $\mathcal{J}_{comp,T}$: set of jobs that are completed at time T ($e_i \leq T$)
  - $\mathcal{J}_{started,T}$: set of jobs running 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)

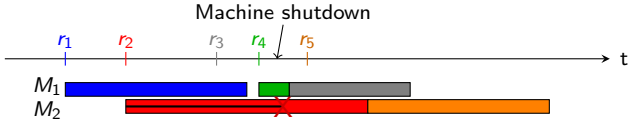


- **Objective function:** Goodput $\Rightarrow$ *effective* utilization, accounts only for tasks that are completed or still running – fraction of useful work up to time T
  - $\mathcal{J}_{comp,T}$: set of jobs that are completed at time T ($e_i \leq T$)
  - $\mathcal{J}_{started,T}$: set of jobs running 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

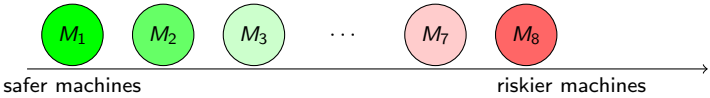
Keep an eye on maximum stretch

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

Algorithms

Main idea:

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

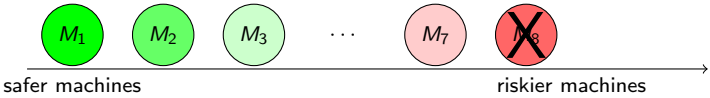


Risk-aware job allocation strategies

Algorithms

Main idea:

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

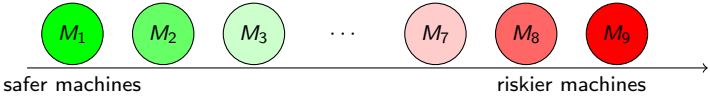


Risk-aware job allocation strategies

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

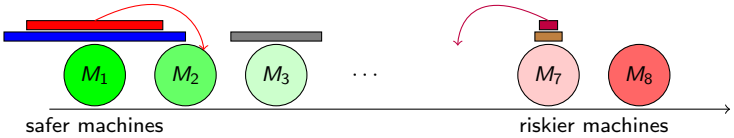
- FirstFitAware (natural approach):
  - Ordered list of machines
  - Jobs mapped to leftmost (safer) machines whenever possible
  - Rightmost (riskier) machines are shutdown whenever necessary
- FirstFitUnaware: Shutdown random machines whenever necessary
- Can we do better than first fit?
  - Interrupting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

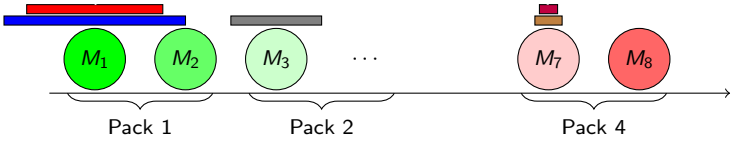
Sophisticated heuristics

- 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

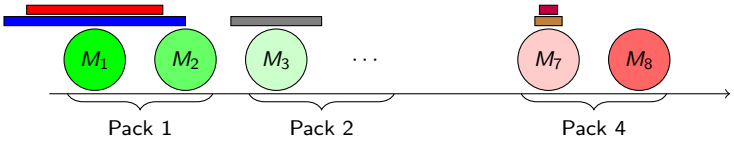
- TargetStretch: Add one queue per machine, target value for max stretch
  potential bad utilization
  No flexibility for mapping to another free machine
- TargetASAP:
  - Start job immediately on target machine or closest machine in neighborhood
  - If not possible, assign on target machine if target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

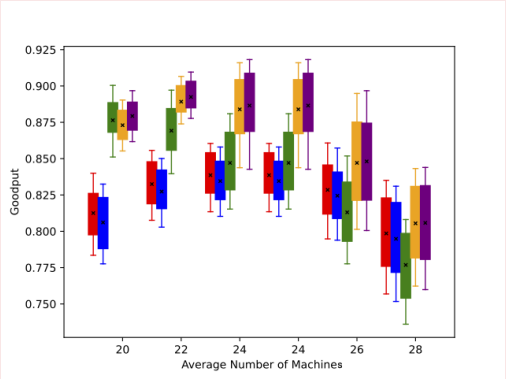
- TargetStretch: Add one queue per machine, target value for max stretch
  potential bad utilization
  No flexibility for mapping to another free machine
- TargetASAP:
  - Start job immediately on target machine or closest machine in neighborhood
  - If not possible, assign on target machine if target stretch not exceeded
  - Other… …table distance)
- Variant P… …ssign jobs to first machines of the pack, to leave machines empty for future large jobs

Technical and kind of painful
despite all simplifying hypotheses ☹



Sophisticated heuristics

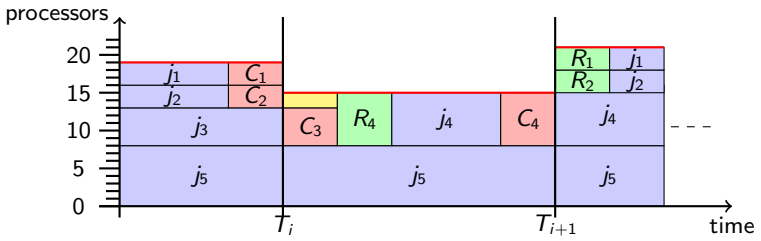
Simulation results using resource variation trace and job traces (Borg)
Significant gains over first-fit algorithms: map the right job to the right machine



Outline

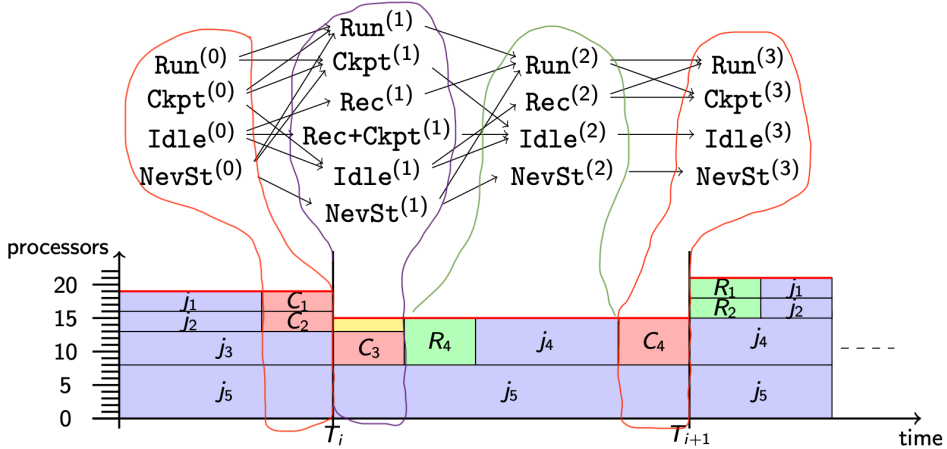
Model

- 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

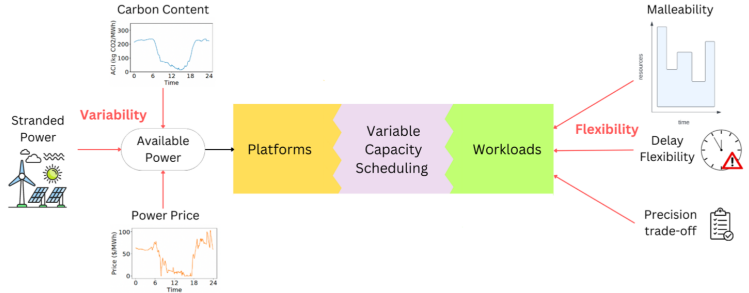


Simulations

- 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

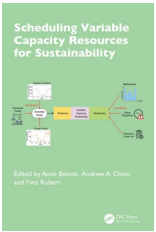
Outline

Conclusion



Many challenging scheduling problems when resources subject to variable capacity ☺

Workshop reports: *Scheduling Variable Capacity Resources for Sustainability*; March 2023, and September 2025, U. Chicago Paris Center



Research directions

Resilience: Consider detectors with non-perfect precision, leading to extra rollbacks due to false alarms; consider failures in variable capacity scheduling problems

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

A few references

- A. Benoit, A. Cavelan, Y. Robert, H. Sun. Assessing General-Purpose Algorithms to Cope with Fail-Stop and Silent Errors. TOPC, 2016
- A. Benoit, T. Herault, Y. Robert, A. Tremodeux. Partial Detectors Versus Replication To Cope With Silent Errors. Euro-Par'25, Dresden, Germany, August 2025.
- L. Perotin, C. Zhang, R. Wijayawardana, A. Benoit, Y. Robert, A. Chien. Risk-Aware Scheduling Algorithms for Variable Capacity Resources. In Proceedings of PMBS, in conjunction with SC'23, Denver, USA, November 2023.
- J. Cendrier, A. Benoit, F. Vivien. Scheduling Jobs Under a Variable Number of Processors. IEEE Transactions on Parallel and Distributed Systems, vol. 37, pp. 427-442, February 2026.
- A. Benoit, A. Chien, Y. Robert, editors. Scheduling Variable Capacity Resources for Sustainability. Chapman and Hall/CRC Press 2026.