Handling **failures** on High Performance Computing platforms: Checkpointing and scheduling techniques

Anne Benoit

LIP, Ecole Normale Supérieure de Lyon, France

Anne.Benoit@ens-lyon.fr

http://graal.ens-lyon.fr/~abenoit/

SBAC-PAD Keynote, November 2-4, 2022

Motivation: 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 ☹

Motivation: Dealing with failures

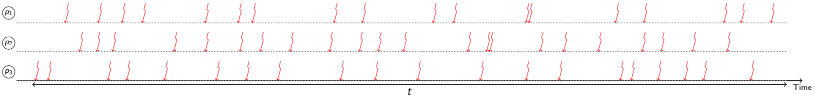
- Consider one processor (e.g. in your laptop)
  - Mean Time Between Failures (MTBF) = 100 years
  - (Almost) no failures in practice ☺

  Why bother about failures?

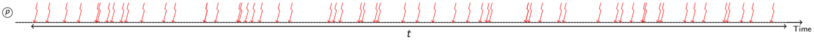
- **Theorem:** The MTBF decreases linearly with the number of processors! With 36500 processors:
  - MTBF = 1 day
  - A failure every day on average!

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 ($\mu_p = \frac{t}{60}$)

Different kind of failures to handle

- 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

Impact of failures

“The internet begins with coal”

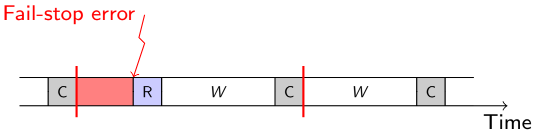
- Nowadays: more than 90 billion kilowatt-hours of electricity a year; requires 34 giant (500 megawatt) coal-powered plants, and produces huge CO_2 emissions
- Explosion of artificial intelligence; AI is hungry for processing power! Need to double data centers in next four years
  → how to get enough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

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

So, how to deal with failures?

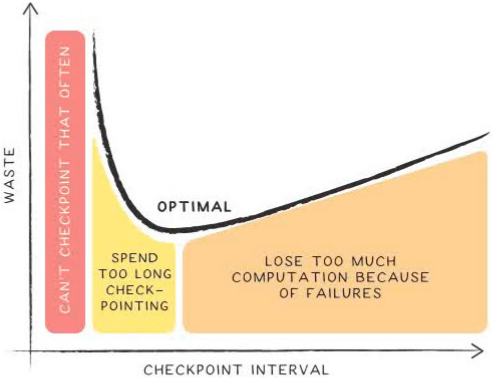
Failures usually handled by adding redundancy:

- Re-execute when a failure strikes (we will come back to this approach in the second part of the talk)
- 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



When should we checkpoint?

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

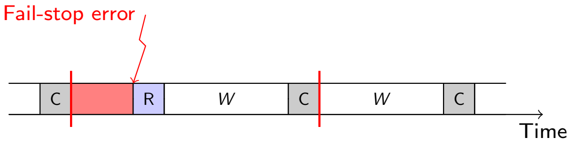


Outline

1. When checkpointing à la Young/Daly is not enough
    - Derivation for Poisson processes
    - Other failure distributions
    - Workflows
2. Resilient scheduling with re-execution
    - Main results for rigid jobs
    - Main results for moldable jobs
    - Simulation results
3. Conclusion

The famous Young/Daly formula

- Periodic checkpointing with period $T = W + C$
- C: Checkpoint time; R: Recovery time
- $\mu_p = \frac{\mu}{p}$: Application MTBF with p processors



Optimal period $W_{YD} = \sqrt{2\mu_p C}$ (Young 1974, Daly 2006)

Outline

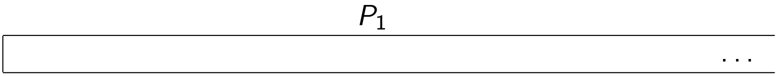
1. When checkpointing à la Young/Daly is not enough
   - Derivation for Poisson processes
   - Other failure distributions
   - Workflows

2. Resilient scheduling with re-execution
   - Main results for rigid jobs
   - Main results for moldable jobs
   - Simulation results

3. Conclusion

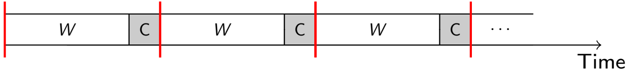
Framework: Poisson processes

- Application with one processor and infinite duration

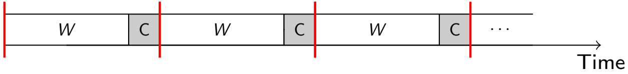


- Failures inter-arrival times: indep. and identically distributed (IID) random variables obeying distribution $\mathcal{D} \sim \textsc{Exp}(\lambda)$
- MTBF $\mathbb{E}(\mathcal{D}) = \mu = \frac{1}{\lambda}$ application/processor MTBF
- Checkpoint, recovery, downtime: cost C, R, D

- Periodic checkpointing with period $T = W + C$



Periodic?



Periodic is optimal when $\mathcal{D} \sim \textsc{Exp}(\lambda)$ (memoryless property)

Optimal period

- $\mathbb{E}(W)$: Expected time to complete a period (of length $W + C$)

$$\mathbb{E}(W) = \mathbb{P}_{succ} \times (W + C) + \mathbb{P}_{fail} \times (\mathbb{E}(T_{lost}) + D + \mathbb{E}(R) + \mathbb{E}(W))$$

$$\boxed{\mathbb{E}(W) = \left(\frac{1}{\lambda} + D\right) e^{\lambda R} \left(e^{\lambda(W+C)} - 1\right)}$$

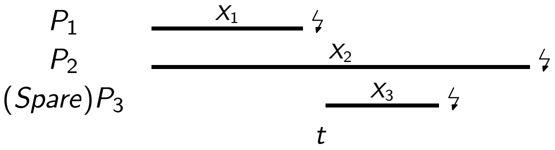
- Find W_{opt} to minimize slowdown $\frac{\mathbb{E}(W)}{W}$

$\boxed{W_{opt} = \frac{1}{\lambda}(\mathbb{L}(-e^{-\lambda C - 1}) + 1)}$ with Lambert function $\mathbb{L}(z) = x \Leftrightarrow z = xe^x$

- When $z \to \frac{-1}{e}$, $\mathbb{L}(z) = -1 + \sqrt{2}y - \frac{2}{3}y^2 + \ldots$, with $y = \sqrt{1 + ez}$

$$Work_{opt} = \sqrt{\frac{2C}{\lambda}} + o(\lambda^{-\frac{1}{2}}) \approx \sqrt{2\mu C}$$

Now with 2 processors



- Two processors, each with failures $X \sim \textsc{Exp}(\lambda)$
- Platform failures:
  - First failure at time $t = \min(X_1, X_2) \sim \textsc{Exp}(2\lambda)$
  - Replace P_1 by fresh spare P_3 (rejuvenate)
  - Second failure still $\sim \textsc{Exp}(2\lambda)$:
    the different history on P_2 and P_3 at time t does not matter (memoryless!)

$$\boxed{\text{Platform failures are IID } \textsc{Exp}(2\lambda)}$$

Now with p processors

$$\boxed{\text{Replace } \lambda \text{ by } \lambda_p = p\lambda \text{ (and } \mu \text{ by } \mu_p = \frac{\mu}{p}\text{), and done ☺}}$$

Why?

- First application failure: minimum of p IID $\textsc{Exp}(\lambda) \sim \textsc{Exp}(p\lambda)$
- When failed processor is replaced (rejuvenation), the history of the other processors does not matter (memoryless!)

$$\boxed{\text{Platform failures are IID } \textsc{Exp}(p\lambda)}$$

Now with p processors and a job of **finite** length W_{job}

- Job of length W_{job}
- Partition into k chunks of length W_i and checkpoint them all ($\sum_{i=1}^{k} W_i = W_{job}$)
- Minimize

$$\mathbb{E}(W_{job}) = e^{\lambda R}\left(\frac{1}{\lambda} + D\right)\sum_{i=1}^{k}(e^{\lambda(W_i+C)} - 1)$$

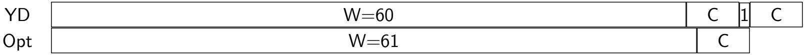
- Solution
  - Same-size chunks by convexity: $W_i = W = \frac{W_{job}}{k}$
  - Differentiate and solve for k with Lambert, find $k_{opt} \in \mathbb{R}$
  - Use either $\max(1, \lfloor k_{opt} \rfloor)$ or $\lceil k_{opt} \rceil$ chunks (whichever leads to minimum)
  - First-order approximation gives $\frac{W_{job}}{k_{opt}} \approx \sqrt{2\mu C}$

Now with p processors and a job of **finite** length W_{job}

- Optimal solution well-understood
- Easy extension when no recovery for first chunk or no checkpoint for last chunk
- Young-Daly is only a first-order approximation

⚠ Young-Daly can significantly differ from optimal for short jobs

Example: $W_{job} = 61,\ W_{YD} = \sqrt{2\mu_p C} = 60,\ C = 5$, final checkpoint



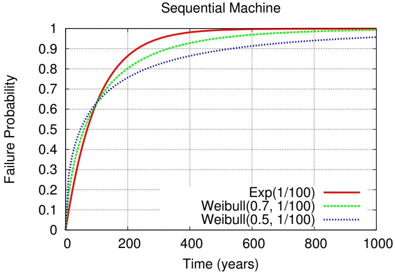
Outline

1. When checkpointing à la Young/Daly is not enough
   - Derivation for Poisson processes
   - Other failure distributions
   - Workflows
2. Resilient scheduling with re-execution
   - Main results for rigid jobs
   - Main results for moldable jobs
   - Simulation results
3. Conclusion

Framework

- What happens if $\mathcal{D}$ is no longer memoryless?
- Processor failures have been shown to obey Weibull or LogNormal distributions...
- Non-constant instantaneous failure rate! ☹

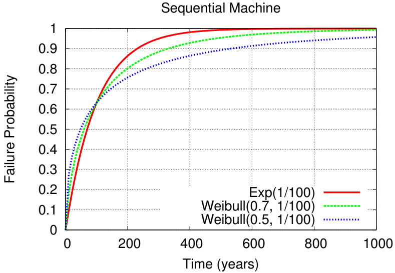
Weibull distribution



$\text{WEIBULL}(k, \lambda)$: Weibull distribution law of shape parameter k and scale parameter λ:

- PDF: $f(t) = k\lambda(t\lambda)^{k-1}e^{-(\lambda t)^k}dt$ for $t \geq 0$
- CDF: $F(t) = 1 - e^{-(\lambda t)^k}$
- Mean $= \frac{1}{\lambda}\Gamma(1 + \frac{1}{k})$

Weibull distribution



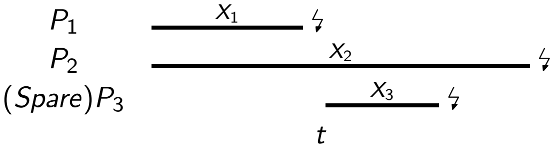
X random variable for $\mathit{Weibull}(k, \lambda)$ failure inter-arrival times:

- If $k < 1$: failure rate decreases with time
  "infant mortality": defective items fail early
- If $k = 1$: $\mathit{Weibull}(1, \lambda) = \mathit{Exp}(\lambda)$ constant failure time

Weibull with 1 processor

- Periodic checkpointing is not optimal:
  if the instantaneous failure rate decreases with time, the length of work chunks (before taking a checkpoint) should increase
- Some dynamic policies have been designed but no closed-form formula ☹
- At least platform failures are IID with 1 processor ☺

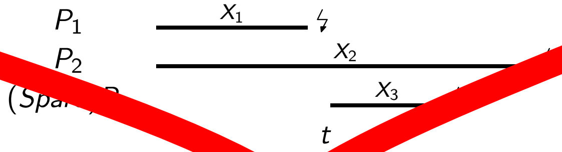
Weibull with 2 processors



- Two processors, each with failures $X \sim \textsc{Weibull}(k, \lambda)$
- Platform:
  - First failure at time $t = \min(X_1, X_2)$ is $\textsc{Weibull}(k, 2\lambda)$
  - Replace P_1 by fresh spare P_3 (rejuvenate)
  - Second failure is not Weibull because of different history on P_2 and P_3 at time t
  - Platform failures are not IID

    ... unless we rejuvenate P_2 together with P_1 after first failure

Weibull with 2 processors



- Two processors, each wi ailur $X \sim \textsc{Weibull}(k, \lambda)$
- Platform:
  - First failure a ime $t = \min(X_1, X_2)$ i $\textsc{Veibull}(k, 2\lambda)$
  - Replace P y fresh spare P_3 (rejuvenate
  - Second ilure is not Weibull because of di ent history on P_2 and
  - P ror

  . . . unle fter st failure

Nobody will rejuvenate 100K processors after each failure

Platform MTBF?

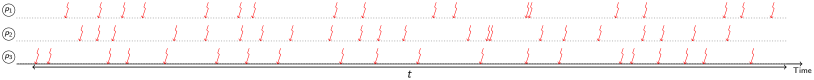
- Rebooting only faulty processor
- Processor failures: IID, obey $\mathcal{D}$ with mean μ
- Platform failures:
  $\Rightarrow$ superposition of p IID processor distributions
  $\Rightarrow$ IID only for Exponential
- Define μ_p by

$$\lim_{F \to +\infty} \frac{F}{n(F)} = \mu_p$$

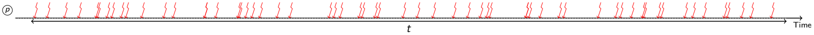
$n(F)$ = number of platform failures until time F is exceeded

Theorem: This limit exists and $\mu_p = \frac{\mu}{p}$ for arbitrary (regular) distributions

Back to Young/Daly



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 ($\mu_p = \frac{t}{60}$)

Since $\mu_p = \frac{\mu}{p}$ for arbitrary (regular) distributions . . .

. . . why not use periodic checkpointing à la Young/Daly $W_{YD} = \sqrt{2\mu_p C}$

. . . and hope for the best?

Accuracy

- Not much known
- Approximations based on computing the waste
- Monte-Carlo simulations (brute force) to compare with optimal period (which is unknown, so binary search all)
- Distance between *periodic* and *optimal*?

State-of-the-art

- Assume constant instantaneous fault rate (after infant mortality and before aging . . .)
- Pretend to rejuvenate all processors at each failure
- Assume that platform failures are Weibull (what are they on each processor?)

Ignore problem and use Young/Daly (with confidence?)

A solution

- Checkpoint parallel jobs under any failure probability distribution
- Dynamic checkpointing strategy
- From one failure to the next!
- After each failure, maximize expected efficiency before the next failure or the end of the job (jobs of finite length)

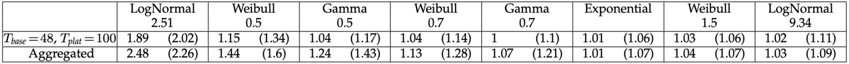
$$\text{Efficiency} = \frac{\text{Work done until next failure}}{\text{Time to next failure}}$$

Technicalities

- Discretization with time quantum
- From one failure to the next, processors keep the same difference in history
  ⇒ NEXT heuristic to optimize efficiency
  ⇒ Dynamic programming in $O(pW^4)$, where W is expressed in quanta
- Asymptotically optimal ☺

At last, a statement about the optimality of the approach for general distributions! ☺ ☺ ☺

Aggregated results



	LogNormal 2.51		Weibull 0.5		Gamma 0.5		Weibull 0.7		Gamma 0.7		Exponential		Weibull 1.5		LogNormal 9.34	
$T_{base} = 48, T_{plat} = 100$	1.89	(2.02)	1.15	(1.34)	1.04	(1.17)	1.04	(1.14)	1	(1.1)	1.01	(1.06)	1.03	(1.06)	1.02	(1.11)
Aggregated	2.48	(2.26)	1.44	(1.6)	1.24	(1.43)	1.13	(1.28)	1.07	(1.21)	1.01	(1.07)	1.04	(1.07)	1.03	(1.09)

Ratio of execution time YoungDaly / NEXT (geom. mean, geom. stdev)

- NEXT always adapts to actual instantaneous failure rate: accounts for the failure history of processors
- Better strategy in all cases
- More significant differences for the realistic distribution laws (LogNormal 2.51 and Weibull 0.5)

Parameters to vary: platform age, job duration, job size, checkpoint duration, individual MTBF

See [Benoit, Perotin, Robert, Vivien. *Checkpointing strategies to protect parallel jobs from non-memoryless fail-stop errors*. Inria RR-9465]

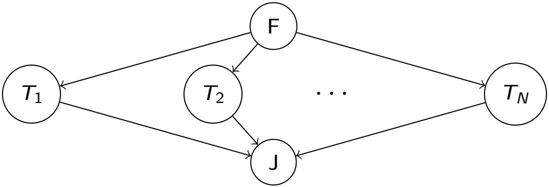
Outline

1. When checkpointing à la Young/Daly is not enough
   - Derivation for Poisson processes
   - Other failure distributions
   - Workflows
2. Resilient scheduling with re-execution
   - Main results for rigid jobs
   - Main results for moldable jobs
   - Simulation results
3. Conclusion

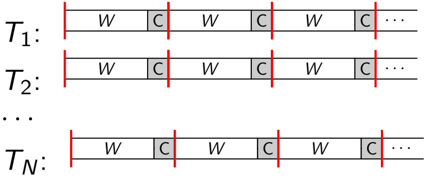
Framework

- Back to memoryless failures ☺
- So far, we have dealt with a tightly-coupled application
- What about a workflow made of several (parallel) tasks?

Fork-join graph



N identical parallel tasks



Optimal Young/Daly period W_{opt} for each task...

Is it good enough?

Parallel tasks

Intuition

- Multiple tasks execute simultaneously
- Higher risk that one of them is severely delayed
  ⇒ Take more checkpoints to mitigate this risk

Solution

- The number of failures of each task follows the *Negative Binomial Distribution*.
- The maximum of n such identical variables is known
  ⇒ Estimation of the number of checkpoints to take

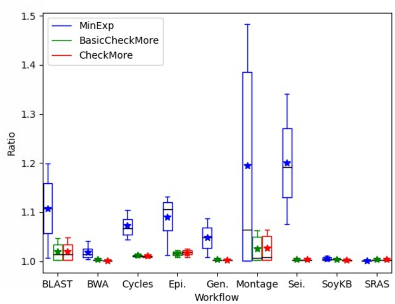
General workflow graphs

Algorithm: CheckMore strategy

- Start with a failure-free schedule $\mathcal{S}$
- Partition it into **virtual slices** with equal-length tasks
- Use previous result on parallel tasks
- Schedule tasks ASAP but keep the initial ordering of $\mathcal{S}$

General workflow graphs

See [Benoit, Perotin, Robert, Sun. *Checkpointing Workflows à la Young/Daly Is Not Good Enough.* ACM TOPC 2022] for evaluation of new strategies



Models needed to assess techniques at scale

without bias ☺

Outline

1. When checkpointing à la Young/Daly is not enough
   - Derivation for Poisson processes
   - Other failure distributions
   - Workflows

2. Resilient scheduling with re-execution
   - Main results for rigid jobs
   - Main results for moldable jobs
   - Simulation results

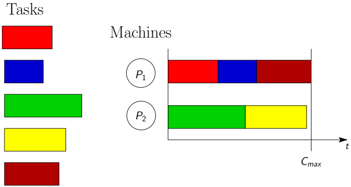
3. Conclusion

Motivation

On large-scale HPC platforms:

- Scheduling parallel jobs is important to improve application performance and system utilization
- Handling job failures is critical as failure/error rates increase dramatically with size of system

We combine job scheduling and failure handling for moldable parallel jobs running on large HPC platforms that are prone to failures



Parallel job models

In the scheduling literature:

- **Rigid jobs**: Processor allocation is fixed by the user and cannot be changed by the system (i.e., fixed, static allocation)
- **Moldable jobs**: Processor allocation is decided by the system but cannot be changed once jobs start execution (i.e., fixed, dynamic allocation)
- **Malleable jobs**: Processor allocation can be dynamically changed by the system during runtime (i.e., variable, dynamic allocation)

We focus on moldable jobs, because:

- They can easily adapt to the amount of available resources (contrarily to rigid jobs)
- They are easy to design/implement (contrarily to malleable jobs)
- Many computational kernels in scientific libraries are provided as moldable jobs

Scheduling model

n moldable jobs to be scheduled on P identical processors

- Job j $(1 \leq j \leq n)$: Choose processor allocation p_j $(1 \leq p_j \leq P)$
- Execution time $t_j(p_j)$ of each job j is a function of p_j
- Area is $a_j(p_j) = p_j \times t_j(p_j)$
- Jobs are subject to arbitrary failure scenarios, which are unknown ahead of time (i.e., semi-online)
- Minimize the makespan (successful completion time of all jobs)

Speedup models

- Roofline model: $t_j(p_j) = \frac{w_j}{\max(p_j, \bar{p}_j)}$, for some $1 \leq \bar{p}_j \leq P$
- Communication model: $t_j(p_j) = \frac{w_j}{p_j} + (p_j - 1)c_j$, where c_j is the communication overhead
- Amdahl's model: $t_j(p_j) = w_j\left(\frac{1-\gamma_j}{p_j} + \gamma_j\right)$, where γ_j is the inherently sequential fraction
- Monotonic model: $t_j(p_j) \geq t_j(p_j + 1)$ and $a_j(p_j) \leq a_j(p_j + 1)$, i.e., execution time non-increasing and area is non-decreasing
- Arbitrary model: $t_j(p_j)$ is an arbitrary function of p_j

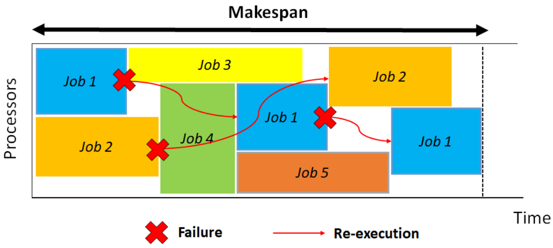
- Rigid jobs: p_j is fixed and hence execution time is t_j

Failure model

- Jobs can fail due to silent errors (or silent data corruptions)
- A lightweight silent error detector (of negligible cost) is available to flag errors at the end of each job's execution
- If a job is hit by silent errors, it must be re-executed (possibly multiple times) till successful completion

A failure scenario $\mathbf{f} = (f_1, f_2, \ldots, f_n)$ describes the number of failures each job experiences during a particular execution

Example: $\mathbf{f} = (2, 1, 0, 0, 0)$ *for an execution of 5 jobs*



Problem complexity

- Scheduling problem clearly NP-hard (failure-free is a special case)
- A scheduling algorithm $\textsc{Alg}$ is said to be a *c-approximation* if its makespan is at most c times that of an optimal scheduler for all possible sets of jobs, and for all possible failure scenarios, i.e.,

$$T_{\textsc{Alg}}(\mathbf{f}, \mathbf{s}) \leq c \times T_{\mathsf{opt}}(\mathbf{f}, \mathbf{s}^*)$$

- $T_{\mathsf{opt}}(\mathbf{f}, \mathbf{s}^*)$ denotes the optimal makespan with scheduling decision $\mathbf{s}^*$ under failure scenario $\mathbf{f}$

Outline

1. When checkpointing à la Young/Daly is not enough
   - Derivation for Poisson processes
   - Other failure distributions
   - Workflows

2. Resilient scheduling with re-execution
   - Main results for rigid jobs
   - Main results for moldable jobs
   - Simulation results

3. Conclusion

Lower bounds

Rigid jobs: p_j is fixed and job j has execution time t_j

Optimal makespan has two lower bounds:

$$T_{\text{opt}}(\mathbf{f}, \mathbf{s}^*) \geq t_{\max}(\mathbf{f})$$
$$T_{\text{opt}}(\mathbf{f}, \mathbf{s}^*) \geq \frac{A(\mathbf{f})}{P}$$

- $t_{\max}(\mathbf{f}) = \max_{j=1\dots n}(f_j + 1) \times t_j$: maximum cumulative execution time of any job under $\mathbf{f}$
- $A(\mathbf{f}) = \sum_{j=1}^{n}(f_j + 1) \times a_j$: total cumulative area

List-based algorithm

Resilient list-based scheduling algorithm, and $O(1)$-approximations for any failure scenario:

- Extends classical batch scheduler that combines reservation and backfilling strategies
- Organizes all jobs in a list (or queue) based on some priority rule
- When a job completes: processors released; if error, inserted back in the queue; remaining jobs scheduled

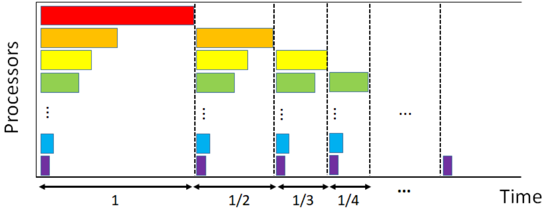
Approximation results:

- 2-approximation using Greedy heuristic without reservation
- 3-approximation using Large Job First priority with reservation

The results nicely extend the ones without job failures [TWY'92: Turek, Wolf, Yu. *Approximate algorithms scheduling parallelizable tasks*. SPAA'92]

Shelf-based algorithm

Resilient shelf-based scheduling heuristic, but $\Omega(\log P)$-approx. for any shelf-based solution in some failure scenario, e.g.:

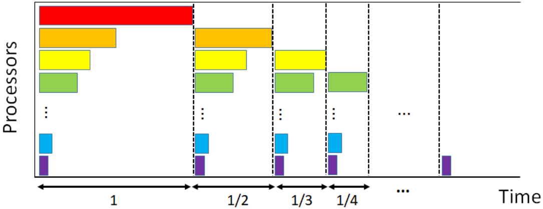


The result defies the $O(1)$-approx. result without failures [TWY'92]

Why not re-execute failed jobs within a same shelf?
Optimal on this example!

Shelf-based algorithm

Resilient shelf-based scheduling heuristic, but $\Omega(\log P)$-approx. for any shelf-based solution in some failure scenario, e.g.:

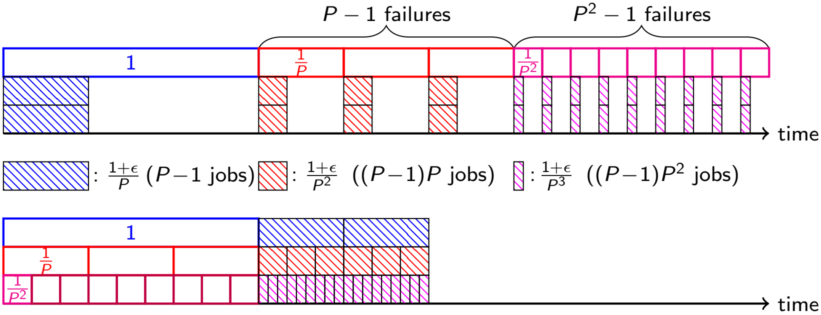


The result defies the $O(1)$-approx. result without failures [TWY'92]

Why not re-execute failed jobs within a same shelf?
Optimal on this example!

Shelf-fill variant: Fill shelfs when error detected

However, there exists a job instance and a failure scenario such that Shelf-fill with the LPT priority rule has an approximation ratio of $\Omega(P)$!


$P-1$ failures
P^2-1 failures
1
$\frac{1}{P}$
$\frac{1}{P^2}$
time
▨ : $\frac{1+\epsilon}{P}$ $(P-1$ jobs)
▨ : $\frac{1+\epsilon}{P^2}$ $((P-1)P$ jobs)
▨ : $\frac{1+\epsilon}{P^3}$ $((P-1)P^2$ jobs)
1
$\frac{1}{P}$
$\frac{1}{P^2}$
time

$+$ *Extensive simulation results of all heuristics using both synthetic jobs and job traces from the Mira supercomputer*, see [Benoit, Le Fèvre, Raghavan, Robert, Sun. *Resilient scheduling heuristics for rigid parallel jobs*. IJNC 2021]

Outline

1. When checkpointing à la Young/Daly is not enough
   - Derivation for Poisson processes
   - Other failure distributions
   - Workflows
2. Resilient scheduling with re-execution
   - Main results for rigid jobs
   - Main results for moldable jobs
   - Simulation results
3. Conclusion

Main results for moldable jobs

Two resilient scheduling algorithms with analysis of approximation ratios and simulation results

1. A list-based scheduling algorithm, called LPA-LIST, and approximation results for several speedup models
2. A batch-based scheduling algorithm, called BATCH-LIST, and approximation result for the arbitrary speedup model
3. Extensive simulations to evaluate and compare (average and worst-case) performance of both algorithms against baseline heuristics

(1) LPA-LIST scheduling algorithm

Two-phase scheduling approach:

- **Phase 1**: Allocate processors to jobs using the Local Processor Allocation (LPA) strategy
  - Minimize a local ratio individually for each job as guided by the property of the LIST scheduling (next slide)
  - The processor allocation p_j will remain unchanged for different execution attempts of the same job j
- **Phase 2**: Schedule jobs with fixed processor allocations using the List Scheduling (LIST) strategy (as in rigid case)
  - Organize all jobs in a list according to any priority order
  - Schedule the jobs one by one at the earliest possible time (with backfilling whenever possible)
  - If a job fails after an execution, insert it back into the queue for rescheduling; Repeat this until the job completes successfully

(1) LPA-LIST scheduling algorithm

Given a processor allocation $\mathbf{p} = (p_1, p_2, \ldots, p_n)$ and a failure scenario $\mathbf{f} = (f_1, f_2, \ldots, f_n)$:

- $A(\mathbf{f}, \mathbf{p}) = \sum_j a_j(p_j)$: total area of all jobs
- $t_{\max}(\mathbf{f}, \mathbf{p}) = \max_j t_j(p_j)$: maximum execution time of any job

Property of LIST Scheduling

For any failure scenario $\mathbf{f}$, if the processor allocation $\mathbf{p}$ satisfies:

$$A(\mathbf{f}, \mathbf{p}) \leq \alpha \cdot A(\mathbf{f}, \mathbf{p}^*) \ ,$$
$$t_{\max}(\mathbf{f}, \mathbf{p}) \leq \beta \cdot t_{\max}(\mathbf{f}, \mathbf{p}^*) \ ,$$

where $\mathbf{p}^*$ is the processor allocation of an optimal schedule, then a LIST schedule using processor allocation $\mathbf{p}$ is $r(\alpha, \beta)$-approximation:

$$r(\alpha, \beta) = \begin{cases} 2\alpha, & \text{if } \alpha \geq \beta \\ \frac{P}{P-1}\alpha + \frac{P-2}{P-1}\beta, & \text{if } \alpha < \beta \end{cases} \tag{1}$$

Eq. (1) is used to guide the local processor allocation (LPA) for each job

(1) Lpa-List scheduling algorithm

Approximation results of Lpa-List for some speedup models:

Speedup Model	Approximation Ratio
Roofline	2
Communication	3[1]
Amdahl	4
Monotonic	$\Theta(\sqrt{P})$

Advantages and disadvantages of Lpa-List:

- **Pros**: Simple to implement, and constant approximation for some common speedup models
- **Cons**: Uncoordinated processor allocation, and high approximation for monotonic/arbitrary model

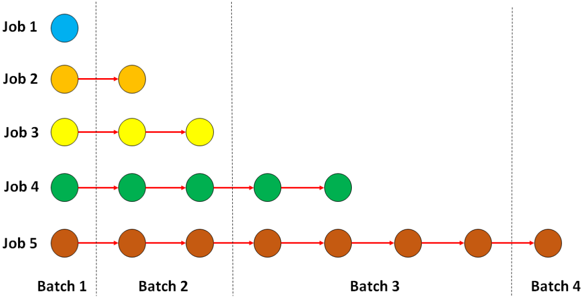
[1] For the communication model, our approx. ratio (3) improves upon the best ratio to date (4), which was obtained without any resilience considerations: [*Havill and Mao. Competitive online scheduling of perfectly malleable jobs with setup times, European Journal of Operational Research, 187:1126–1142, 2008*]

(2) BATCH-LIST scheduling algorithm

Batched scheduling approach:

- Different execution attempts of the jobs are organized in batches that are executed one after another
- In each batch k $(= 1, 2, \dots)$, all pending jobs are executed a maximum of 2^{k-1} times
- Uncompleted jobs in each batch will be processed in the next batch

Example: an execution of 5 jobs under a failure scenario $\mathbf{f} = (0, 1, 2, 4, 7)$



(2) Batch-List scheduling algorithm

Within each batch k:

- Processor allocations are done for pending jobs using the Mt-Allotment algorithm[2], which guarantees near optimal allocation (within a factor of $1 + \epsilon$)
- The maximum of 2^{k-1} execution attempts of the pending jobs are scheduling using the List strategy

Approximation Result of Batch-List

The Batch-List algorithm is $\Theta((1+\epsilon)\log_2(f_{\max}))$-approximation for arbitrary speedup model, where $f_{\max} = \max_j f_j$ is the maximum number of failures of any job in a failure scenario

[2] The algorithm has runtime polynomial in $1/\epsilon$ and works for jobs in SP-graphs/trees (of which a set of independent linear chains is a special case) [*Lepère, Trystram, and Woeginger. Approximation algorithms for scheduling malleable tasks under precedence constraints. European Symposium on Algorithms, 2001*]

Outline

1. When checkpointing à la Young/Daly is not enough
   - Derivation for Poisson processes
   - Other failure distributions
   - Workflows
2. Resilient scheduling with re-execution
   - Main results for rigid jobs
   - Main results for moldable jobs
   - Simulation results
3. Conclusion

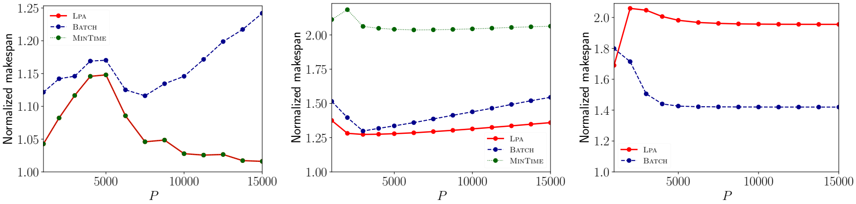
Performance evaluation

We evaluate the performance of our algorithms using simulations

- Synthetic jobs under three speedup models (Roofline, Communication, Amdahl) and different parameter settings
- Job failures follow exponential distribution with varying error rate λ
- Baseline algorithm for comparison:
  - MINTIME: allocate processors to minimize execution time of each job and schedule jobs using LIST
- Priority rules used in LIST:
  - LPT (Longest Processing Time)
- Results normalized by a lower bound (minimum possible total execution time of a job, minimum possible total area)

Simulation results — with varying number of processors P

- In Roofline model, LPA (and MINTIME) has better performance, thanks to it simple and effective local processor allocation strategy
- In Communication model, BATCH catches up with LPA and performs better than MINTIME
- In Amdahl's model (where parallelizing a job becomes less efficient due to extra communication overhead), BATCH has the best performance, thanks to its coordinated processor allocation



(a) Roofline model

(b) Communication model

(c) Amdahl's model

Simulations — Summary of perf of three algos (over loose bound)

- Both algorithms (LPA and BATCH) perform significantly better than the baseline MINTIME
- Over the whole set of simulations, our best algorithm (LPA or BATCH) is within a factor of 1.47 of the lower bound on average, and within a factor of 1.8 of the lower bound in the worst case

Speedup model		**Roofline**	**Communication**	**Amdahl**
LPA	Expected	1.055	1.310	1.960
	Maximum	1.148	1.379	2.059
BATCH	Expected	1.154	1.430	**1.465**
	Maximum	1.280	1.897	**1.799**
MINTIME	Expected	1.055	2.040	14.412
	Maximum	1.148	2.184	24.813

- See [Benoit, Le Fèvre, Perotin, Raghavan, Robert, Sun. *Resilient scheduling of moldable jobs on failure-prone platforms*. Cluster 2020] and [Benoit, Le Fèvre, Perotin, Raghavan, Robert, Sun. *Resilient scheduling of moldable parallel jobs to cope with silent errors*. IEEE TC 2021] for detailed results.

Outline

1. When checkpointing à la Young/Daly is not enough
   - Derivation for Poisson processes
   - Other failure distributions
   - Workflows

2. Resilient scheduling with re-execution
   - Main results for rigid jobs
   - Main results for moldable jobs
   - Simulation results

3. Conclusion

Conclusion

Take-aways:

- Future HPC platforms demand simultaneous resource scheduling and resilience considerations for parallel applications
- Young/Daly formula commonly used to determine the optimal checkpointing period, but it is not always the best strategy
- Resilient scheduling algorithms for rigid and moldable parallel jobs with provable performance guarantees and good performance

Future work:

- Still a lot of challenges to address, and techniques to be developed for many kinds of high-performance applications, in order to handle failures, using checkpointing, re-execution, and also replication
- In particular, life is more complicated with non-memoryless failure distributions and general workflow applications!

Thanks!!! And have a great time in Bordeaux!