Revisiting checkpointing techniques and I/O bandwidth-sharing strategies for HPC platforms

Anne Benoit

LIP, Ecole Normale Supérieure de Lyon, France

Anne.Benoit@ens-lyon.fr

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

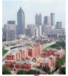
LIP Seminar, May 10, 2023

A brief word about myself

Grenoble, France
1995-1997: Math studies
1997-2000: ENSIMAG Engineer school
Systems and Networks
2000-2003: PhD thesis
Performance evaluation, Markov chains

Edinburgh, UK
2003-2005: Post-doc
Algorithmic skeletons

2009-2014: Junior member of IUF



ENS Lyon, France
2005-Present: Associate Prof.
Multi-criteria scheduling, resilience, energy, memory, I/Os, ...

2009: Habilitation (HDR)
Scheduling pipelined applications

Julie
2012

Sophie
2014

Georgia Tech, Atlanta, USA
2017-2018: Visiting Ass. Prof.
Scheduling, partitioning, data locality

Pedagogical responsibilities @ ENSL: L3 (2006—2010; 2018—2022); Master (2015-2017); Department Chair (2022— Pres)

Major scientific responsibilities: AE in Chief of JPDC & Parco
IEEE TCPP Chair since 2020, IPDPS'22 General Chair; Steering Committees
Program Chair for HiPC'16, ICPP'17, SC'17, IPDPS'18

1 book, 53 journal publications, 104 conference publications

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

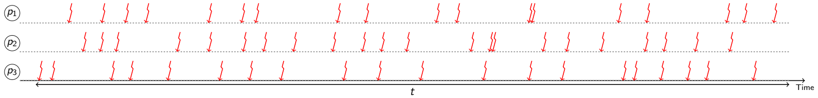
- Consider one processor (e.g. in your laptop)
  - Mean Time Between Failures (MTBF) = 100 years
  - (Almost) no failures in practice ☺

  Why bother about failures?

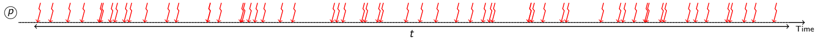
- **Theorem:** The MTBF decreases linearly with the number of processors! With 36500 processors:
  - MTBF = 1 day
  - A failure every day on average!

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

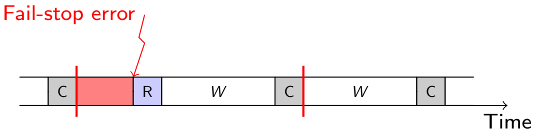
Different kind of failures to handle

- Fail-stop errors, a.k.a. failures:
  - 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

So, how to deal with failures?

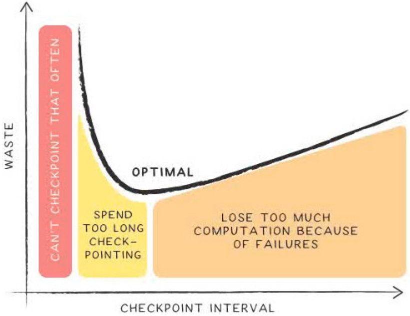
Failures usually handled by adding redundancy:

- Re-execute when a failure strikes (we may loose a lot of work at each failure)
- Replicate the work (for instance, use only half of the processors, and the other half is used to redo the same computation – waste of resources?)
- 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?



Optimal checkpointing period well understood in theory, but we need to revisit it in some real-world settings

Another practical problem: Checkpoint contention

Context:

- Several applications running simultaneously on an HPC platform
- The applications post concurrent I/O operations, for instance checkpoints (but works for *any I/O operations*)
- Demands exceed total available I/O bandwidth

Question:

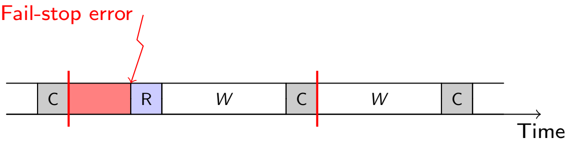
- What is the best way to share the bandwidth between applications?

State-of-the-art strategies are far from optimal!

Outline

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)

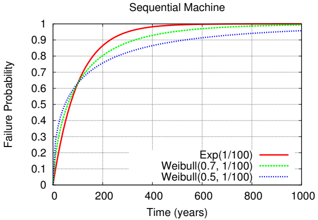
Well-understood for memoryless distributions (Exp for instance)

Outline

Framework: Non memoryless distributions

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

WEIBULL(k, λ): Weibull distribution law of shape parameter k and scale parameter λ

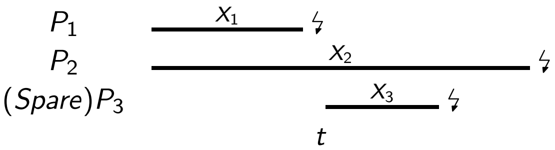


- If $k < 1$: failure rate decreases with time
  "infant mortality": defective items fail early
- If $k = 1$: $\mathit{Weibull}(1, \lambda) = \mathit{Exp}(\lambda)$ constant failure rate

Weibull with one 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 there are no closed-form formula ☹
- At least, platform failures are IID with one processor ☺

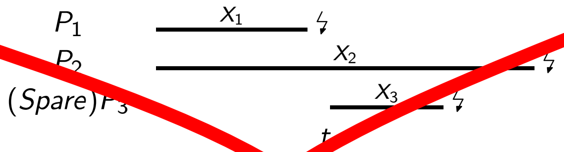
Weibull with two processors



- 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 two 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
  - Platfor

    … unle … after first failure

Nobody will rejuvenate 100K processors after each failure

Platform MTBF?

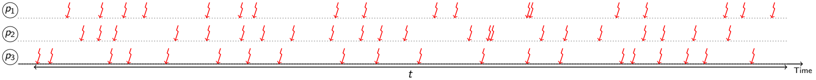
- Rebooting only faulty processor
- Processor failures: IID, obey $\mathcal{D}$ with mean μ
- Platform failures:
  $\Rightarrow$ superposition of p IID processor distributions
  $\Rightarrow$ IID only for Exponential
- Define μ_p by

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

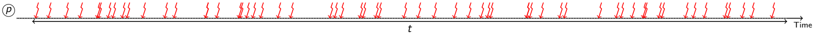
$n(F)$ = number of platform failures until time F is exceeded

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?

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?)

How far is this periodic checkpoint strategy from optimal?

A solution

- Problem: Checkpoint parallel jobs under any failure probability distribution, for an efficient execution
- Solution: Dynamic checkpointing strategy – Take decisions 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
  $\Rightarrow$ NEXT heuristic to optimize efficiency
  $\Rightarrow$ 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! ☺ ☺ ☺

How does it work in practice?

Aggregated results (the higher the better):
Ratio of execution time YoungDaly / NEXT (geom. mean, geom. stdev)

	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)

- 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

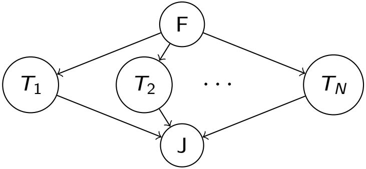
See [Benoit, Perotin, Robert, Vivien. *Checkpointing strategies to protect parallel jobs from non-memoryless fail-stop errors*. Inria RR-9465, 2022. Under revision at TOPC, `https://inria.hal.science/hal-03610883v2`]

Outline

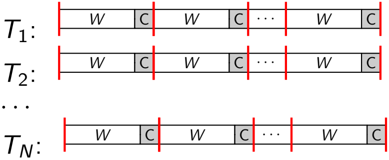
Framework

- 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

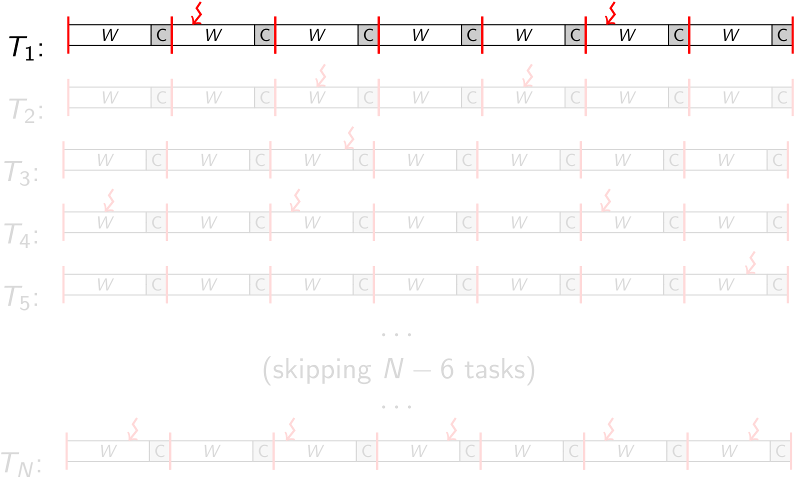


N identical parallel tasks

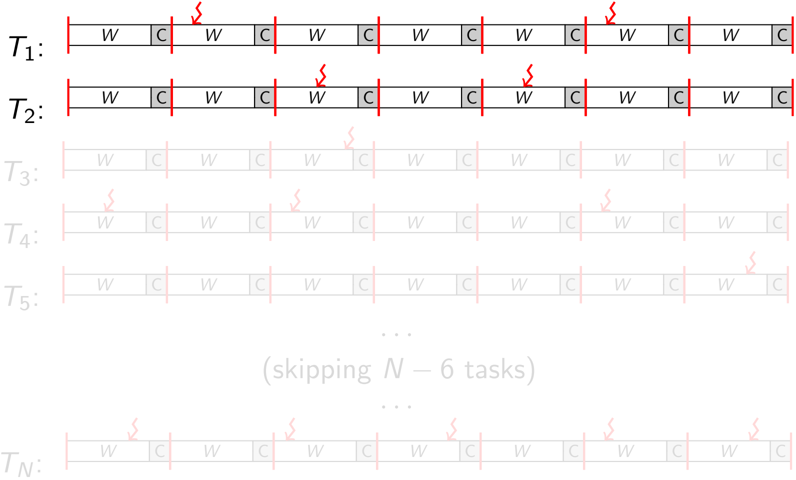


Optimal Young/Daly period W_{opt} for each task...
Is it good enough?

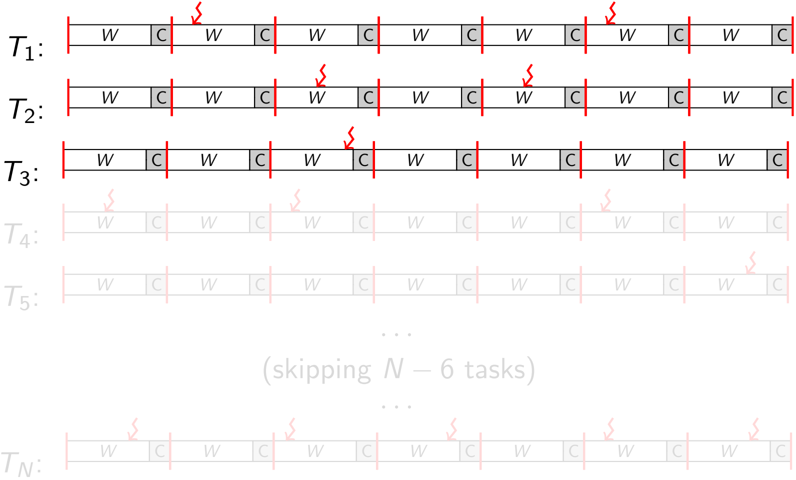
Example with N identical tasks



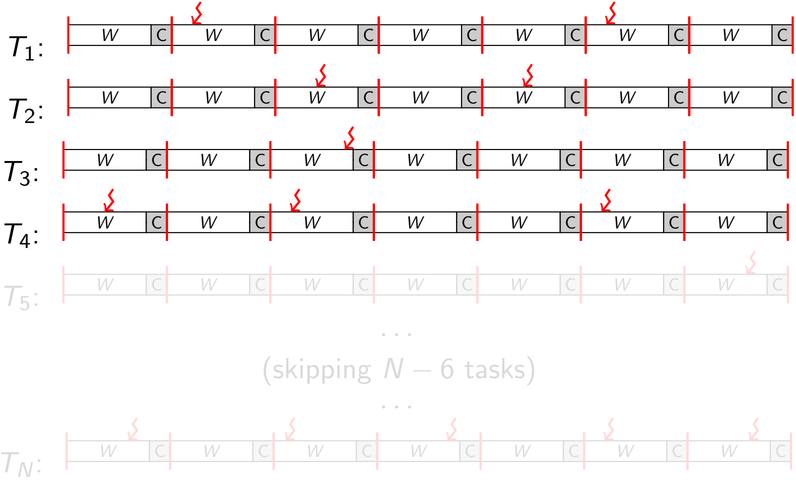
Example with N identical tasks



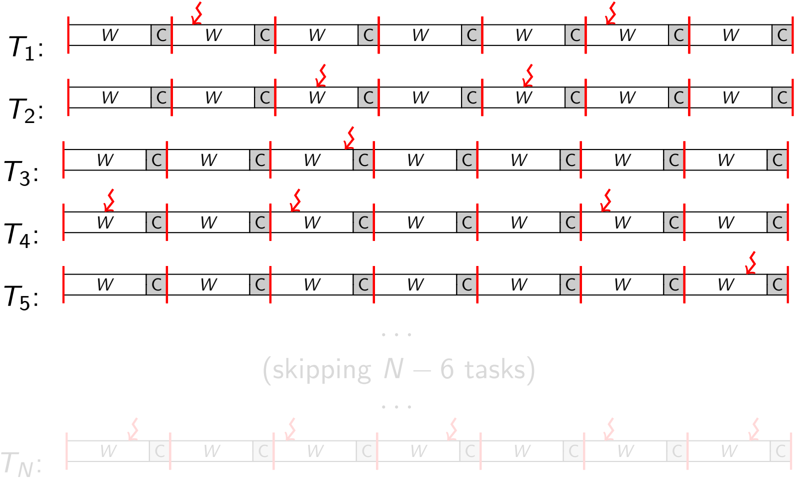
Example with N identical tasks



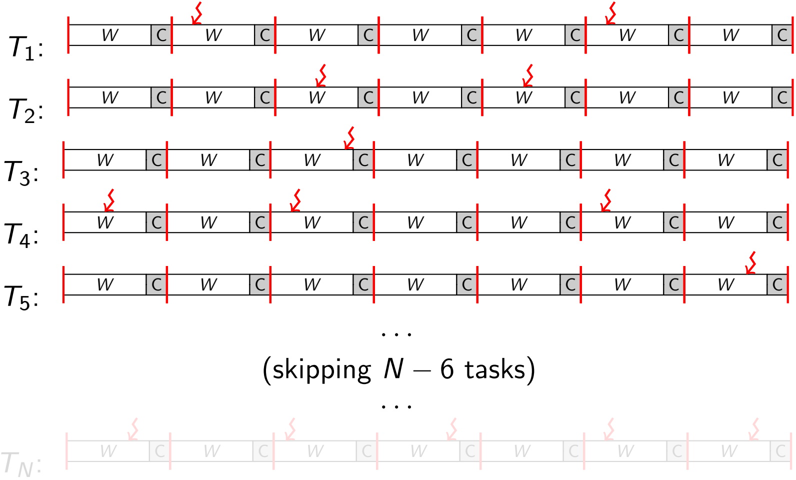
Example with N identical tasks



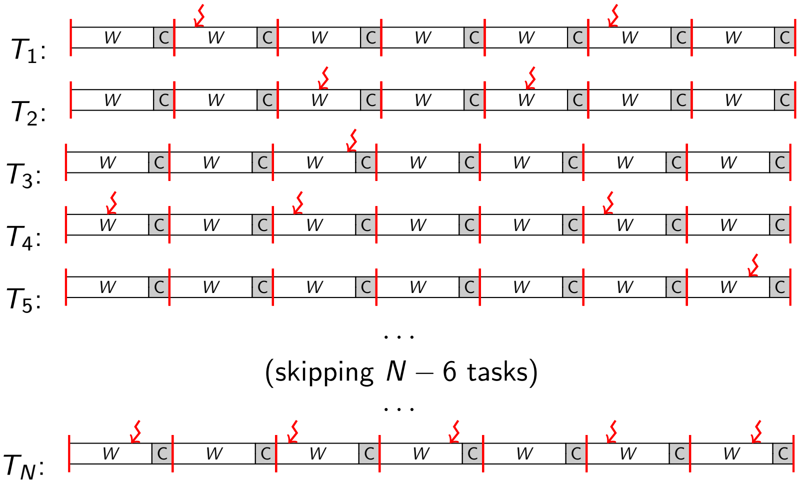
Example with N identical tasks



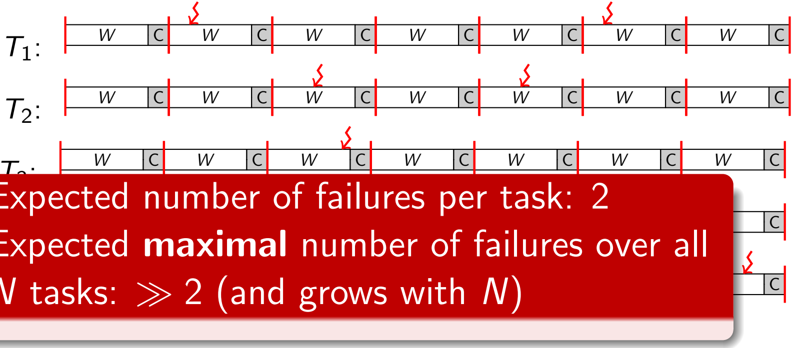
Example with N identical tasks



Example with N identical tasks



Example with N identical tasks

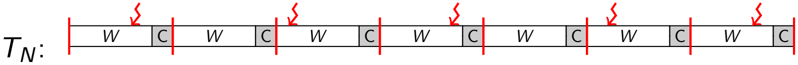


Expected number of failures per task: 2

Expected **maximal** number of failures over all N tasks: $\gg 2$ (and grows with N)

(skipping $N-6$ tasks)

...



Parallel tasks

Intuition

- Multiple tasks execute simultaneously
- Higher risk that one of them is severely delayed
  $\Rightarrow$ 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
  $\Rightarrow$ Estimation of the number of checkpoints to take

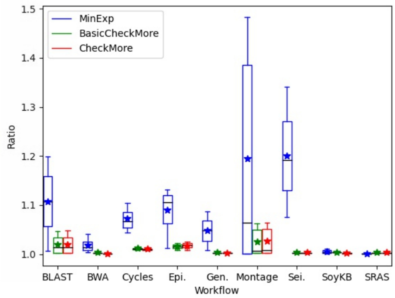
General workflow graphs

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

Problem overview

Context: Applications posting concurrent I/O operations; how to *best* share the bandwidth?

- What objective(s) function(s)?
- How to assess *only* the impact of bandwidth-sharing strategies (BwSS)?

Interplay with batch scheduling: A chicken-and-egg problem

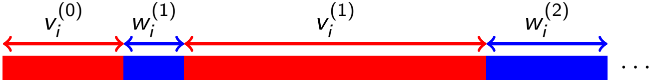
- Change in BwSS impacts application completion times, which impacts opportunities for the batch scheduler, which impacts opportunities for BwSS

The solution

- Study performance in a window $[T_{begin}, T_{end}]$ during which no application can start nor complete

Framework

- A set of m applications, $\mathcal{A}_1, \ldots, \mathcal{A}_m$, released at times $\tau_1, \ldots, \tau_m$
- Each application $\mathcal{A}_i$ executes an alternating sequence of work phases and I/O operations:



  - $v_i^{(j)}$ volume of j-th I/O: known when I/O is posted
  - $w_i^{(j)}$ duration of j-th work: not known until it terminates
- Application $\mathcal{A}_i$ uses p_i nodes
- Total bandwidth B; node bandwidth b; $b_i = \min(B, p_i b)$ is the max bandwidth that can be granted to $\mathcal{A}_i$
- Bandwidth allocation changes whenever an I/O is posted, an I/O completes, or the I/O scheduler triggers an event

Objectives

Main objective function: MinYield (ratio actual progress / ideal progress)

- Yield of $\mathcal{A}_i$ at time t: $y_i(t) = \dfrac{W_i^{(done)}(t) + \frac{V_i^{(transferred)}(t)}{b_i}}{t - \tau_i}$
- MinYield: Maximize minimum yield at the end of the window: $\textsc{Maximize}\, \min_{1 \le i \le m} y_i(T_{end})$

Other objective functions: Maximize platform utilization or sum of actual progress of applications

- Utilization: $\textsc{Maximize} \dfrac{\sum_{1 \le i}^{m} p_i \left(W_i^{(done)}(T_{end}) - W_i^{(done)}(T_{begin}) \right)}{(T_{end} - T_{begin}) \sum_{1 \le i}^{m} p_i}$
- Efficiency:

$$\textsc{Maximize} \frac{\sum_{1 \le i}^{m} p_i \left(W_i^{(done)}(T_{end}) - W_i^{(done)}(T_{begin}) + \frac{V_i^{(transferred)}(T_{end}) - V_i^{(transferred)}(T_{begin})}{b_i} \right)}{(T_{end} - T_{begin}) \sum_{1 \le i}^{m} p_i}$$

Outline

1. When checkpointing à la Young/Daly is not enough
   - With arbitrary failure distributions
   - For workflows
2. Revisiting I/O bandwidth-sharing strategies
   - **Bandwidth-sharing strategies**
   - Lower bounds on competitive ratios
   - Performance of strategies in practice
3. Conclusion

Greedy strategies

- FAIRSHARE: app. $\mathcal{A}_i$ is allocated a bandwidth $\min\left(1, \frac{B}{\sum_j b_j}\right) b_i$
- FCFS: greedily allocate the bandwidth to applications sorted by non-decreasing R_i (time when last I/O operation was posted)
- GREEDYCOM: greedily allocate the bandwidth to applications sorted by non-decreasing ratio $\mathcal{V}_i / b_i$, i.e., by remaining time to complete the pending I/O (priority to short coms)
- GREEDYYIELD: greedily allocate the bandwidth to applications sorted by non-decreasing yields $y_i(t)$
- PERIODICGREEDYYIELD (δ): GREEDYYIELD + events triggered every δ seconds. $\delta = \frac{T_{end} - T_{begin}}{2\#I/O \text{ in } [T_{begin}, T_{end}]}$
- LOOKAHEADGREEDYYIELD: for each $\mathcal{A}_i$, compute the minimum yield Z_i that can be achieved if $\mathcal{A}_i$ is given priority and allocated its maximum bandwidth b_i, and where the remaining bandwidth $B - b_i$ is allocated following GREEDYYIELD for the other applications

Greedy strategies

- FAIRSHARE: app. $\mathcal{A}_i$ is allocated a bandwidth $\min\left(1, \frac{B}{\sum_j b_j}\right) b_i$
- FCFS: greedily allocate the bandwidth to applications sorted by non-decreasing R_i (time when last I/O operation was posted)
- GREEDYCOM: greedily allocate the bandwidth to applications sorted by non-decreasing ratio $\mathcal{V}_i/b_i$, i.e., by remaining time to complete the pending I/O (priority to short coms)
- GREEDYYIELD: greedily allocate the bandwidth to applications sorted by non-decreasing yields $y_i(t)$
- PERIODICGREEDYYIELD (δ): GREEDYYIELD + events triggered every δ seconds. $\delta = \frac{T_{end} - T_{begin}}{2\#I/O \text{ in } [T_{begin}, T_{end}]}$
- LOOKAHEADGREEDYYIELD: for each $\mathcal{A}_i$, compute the minimum yield Z_i that can be achieved if $\mathcal{A}_i$ is given priority and allocated its maximum bandwidth b_i, and where the remaining bandwidth $B - b_i$ is allocated following GREEDYYIELD for the other applications

Greedy strategies

- FAIRSHARE: app. $\mathcal{A}_i$ is allocated a bandwidth $\min\left(1, \frac{B}{\sum_j b_j}\right) b_i$
- FCFS: greedily allocate the bandwidth to applications sorted by non-decreasing R_i (time when last I/O operation was posted)
- GREEDYCOM: greedily allocate the bandwidth to applications sorted by non-decreasing ratio $\mathcal{V}_i / b_i$, i.e., by remaining time to complete the pending I/O (priority to short coms)
- GREEDYYIELD: greedily allocate the bandwidth to applications sorted by non-decreasing yields $y_i(t)$
- PERIODICGREEDYYIELD (δ): GREEDYYIELD + events triggered every δ seconds. $\delta = \frac{T_{end} - T_{begin}}{2\#I/O \text{ in } [T_{begin}, T_{end}]}$
- LOOKAHEADGREEDYYIELD: for each $\mathcal{A}_i$, compute the minimum yield Z_i that can be achieved if $\mathcal{A}_i$ is given priority and allocated its maximum bandwidth b_i, and where the remaining bandwidth $B - b_i$ is allocated following GREEDYYIELD for the other applications

More involved strategies

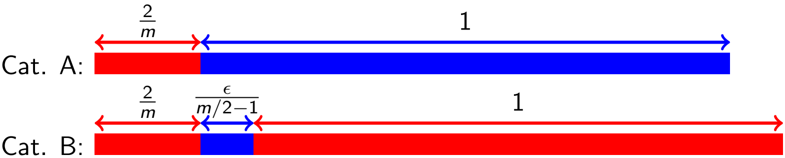
- SET-10 strategy [Boito et al., 2022]
  - Estimates average iteration length for each application (hoping that applications are periodic)
  - Partition apps according to these lengths, and grant bandwidth to a single application per set (FCFS)
- BESTNEXTEV strategy [Benoit et al., 2023]
  - Sophisticated algorithm partitioning the interval of remaining time, and find next *predictable* event (not the I/O arrival), where the min yield is maximized
  - Strategy to optimally compute bandwidth allocation maximizing the minimum yield at a given time t
  - Need to partition the interval and search for events

Outline

1. When checkpointing à la Young/Daly is not enough
   - With arbitrary failure distributions
   - For workflows
2. Revisiting I/O bandwidth-sharing strategies
   - Bandwidth-sharing strategies
   - Lower bounds on competitive ratios
   - Performance of strategies in practice
3. Conclusion

Competitive ratios

- A strategy $\mathcal{S}$ has a competitive ratio ρ for $\textsc{Obj}$ (to be maximized) if, for any instance $\mathcal{I}$, $\textsc{Obj}(\mathcal{S},\mathcal{I}) \times \rho \geq \textsc{Obj}(\textsc{Optimal},\mathcal{I})$
- Lower bound: provide an example with an instance s.t. $\textsc{Obj}(\mathcal{S},\mathcal{I}) \times \rho_{lb} < \textsc{Obj}(\textsc{Optimal},\mathcal{I})$
- Example, with window $[T_{begin}, T_{end}] = [0,1]$; m applications released at time 0 (with $m \geq 4$ and m even); All applications satisfy $b_i = B = 1$ and $p_i = 1$
- Two categories of applications, $\frac{m}{2}$ applications of each type:



Competitive ratio example

- Category A: $\frac{m}{2}$ applications

- Category B: $\frac{m}{2}$ applications

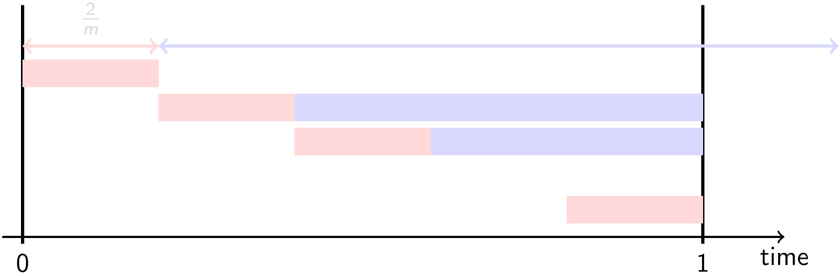
Total requested I/O volume at time 0: 2

Total bandwidth $= 1$: at most $\frac{m}{2}$ applications can complete their first I/O operation by time 1

Best case for utilization and efficiency: $\frac{m}{2}$ applications can complete their first I/O operation by time 1: which ones?

Competitive ratio example – Cat. A

I/Os of Category A applications are completed

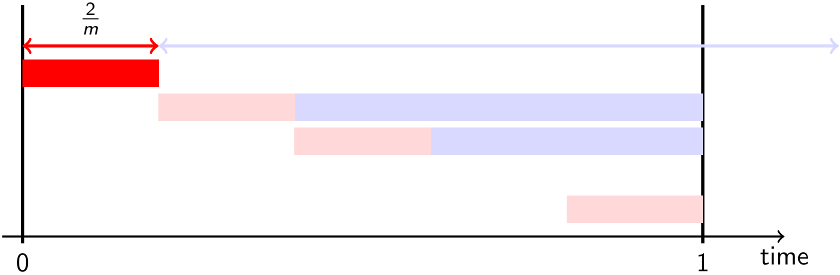


$$\text{Efficiency}_A = \frac{\sum_{i \in A \cup B}(V_i^{(transferred)} + W_i^{(done)})}{m} = \frac{\sum_{i=1}^{m/2}(\frac{2}{m} + \frac{m-2i}{m})}{m} = \frac{m+2}{4m}$$

$$\text{Utilization}_A = \frac{\sum_{i \in A \cup B} W_i^{(done)}}{m} = \frac{\sum_{i=1}^{m/2} \frac{m-2i}{m}}{m} = \frac{m-2}{4m}$$

Competitive ratio example – Cat. A

I/Os of Category A applications are completed

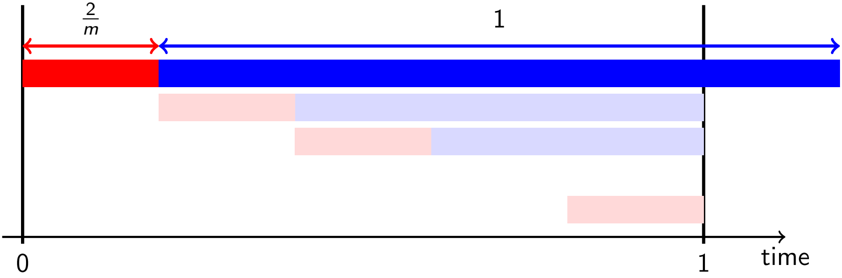


$$\text{EFFICIENCY}_A = \frac{\sum_{i \in A \cup B} (V_i^{(transferred)} + W_i^{(done)})}{m} = \frac{\sum_{i=1}^{m/2} (\frac{2}{m} + \frac{m-2i}{m})}{m} = \frac{m+2}{4m}$$

$$\text{UTILIZATION}_A = \frac{\sum_{i \in A \cup B} W_i^{(done)}}{m} = \frac{\sum_{i=1}^{m/2} \frac{m-2i}{m}}{m} = \frac{m-2}{4m}$$

Competitive ratio example – Cat. A

I/Os of Category A applications are completed

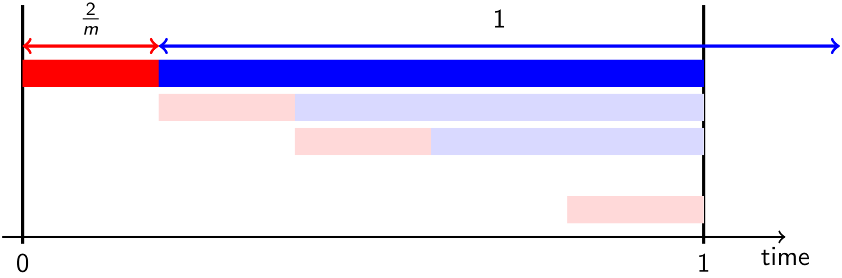


$$\text{EFFICIENCY}_A = \frac{\sum_{i \in A \cup B}(V_i^{(transferred)} + W_i^{(done)})}{m} = \frac{\sum_{i=1}^{m/2}(\frac{2}{m} + \frac{m-2i}{m})}{m} = \frac{m+2}{4m}$$

$$\text{UTILIZATION}_A = \frac{\sum_{i \in A \cup B} W_i^{(done)}}{m} = \frac{\sum_{i=1}^{m/2} \frac{m-2i}{m}}{m} = \frac{m-2}{4m}$$

Competitive ratio example – Cat. A

I/Os of Category A applications are completed

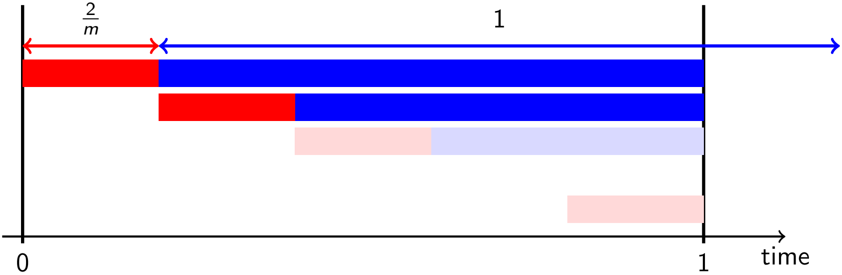


$$\textsc{Efficiency}_A = \frac{\sum_{i \in A \cup B}(V_i^{(transferred)} + W_i^{(done)})}{m} = \frac{\sum_{i=1}^{m/2}(\frac{2}{m} + \frac{m-2i}{m})}{m} = \frac{m+2}{4m}$$

$$\textsc{Utilization}_A = \frac{\sum_{i \in A \cup B} W_i^{(done)}}{m} = \frac{\sum_{i=1}^{m/2} \frac{m-2i}{m}}{m} = \frac{m-2}{4m}$$

Competitive ratio example – Cat. A

I/Os of Category A applications are completed

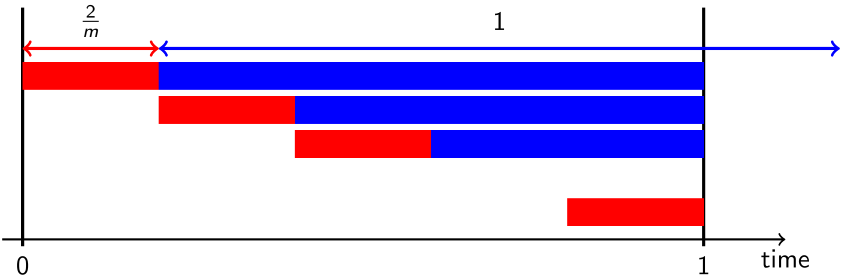


$$\text{EFFICIENCY}_A = \frac{\sum_{i \in A \cup B} (V_i^{(transferred)} + W_i^{(done)})}{m} = \frac{\sum_{i=1}^{m/2} (\frac{2}{m} + \frac{m-2i}{m})}{m} = \frac{m+2}{4m}$$

$$\text{UTILIZATION}_A = \frac{\sum_{i \in A \cup B} W_i^{(done)}}{m} = \frac{\sum_{i=1}^{m/2} \frac{m-2i}{m}}{m} = \frac{m-2}{4m}$$

Competitive ratio example – Cat. A

I/Os of Category A applications are completed

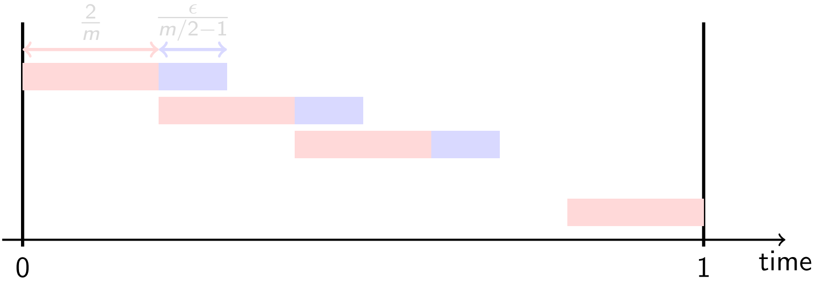


$$\textsc{Efficiency}_A = \frac{\sum_{i \in A \cup B} (V_i^{(transferred)} + W_i^{(done)})}{m} = \frac{\sum_{i=1}^{m/2} (\frac{2}{m} + \frac{m-2i}{m})}{m} = \frac{m+2}{4m}$$

$$\textsc{Utilization}_A = \frac{\sum_{i \in A \cup B} W_i^{(done)}}{m} = \frac{\sum_{i=1}^{m/2} \frac{m-2i}{m}}{m} = \frac{m-2}{4m}$$

Competitive ratio example – Cat. B

I/Os of Category B applications are completed



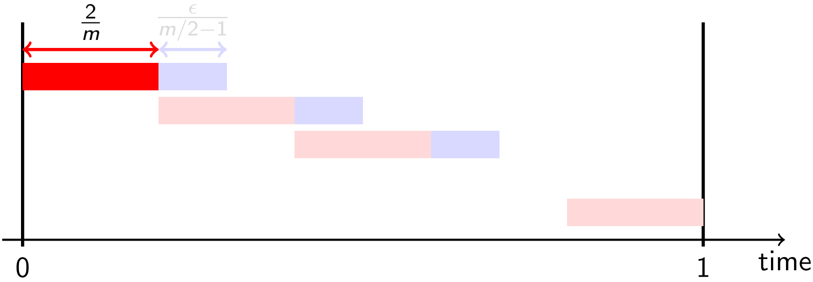
$$\textsc{Efficiency}_B = \frac{\sum_{i \in A \cup B} (V_i^{(transferred)} + W_i^{(done)})}{m} = \frac{\sum_{i=1}^{m/2} (\frac{2}{m} + \frac{\epsilon}{m/2-1})}{m} = \frac{1+\epsilon}{m}$$

$$\textsc{Utilization}_B = \frac{\sum_{i \in A \cup B} W_i^{(done)}}{m} = \frac{\epsilon}{m}$$

... *Almost no work is done!*

Competitive ratio example – Cat. B

I/Os of Category B applications are completed



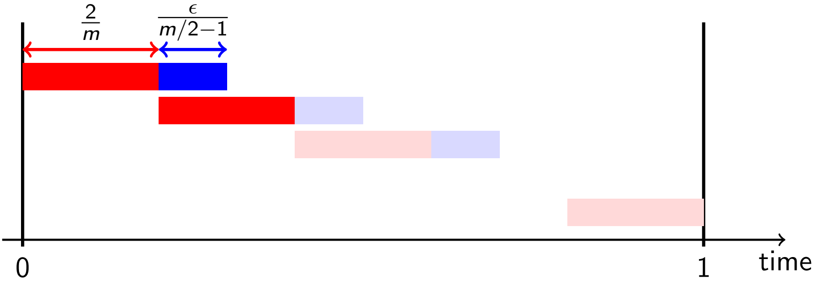
$$\textsc{Efficiency}_B = \frac{\sum_{i \in A \cup B} (V_i^{(transferred)} + W_i^{(done)})}{m} = \frac{\sum_{i=1}^{m/2} (\frac{2}{m} + \frac{\epsilon}{m/2-1})}{m} = \frac{1+\epsilon}{m}$$

$$\textsc{Utilization}_B = \frac{\sum_{i \in A \cup B} W_i^{(done)}}{m} = \frac{\epsilon}{m}$$

... *Almost no work is done!*

Competitive ratio example – Cat. B

I/Os of Category B applications are completed



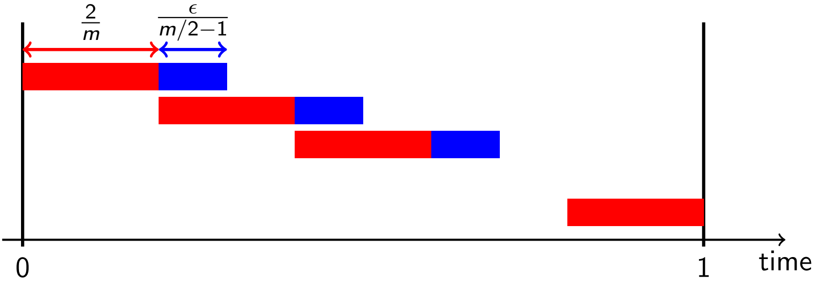
$$\text{EFFICIENCY}_B = \frac{\sum_{i \in A \cup B} (V_i^{(transferred)} + W_i^{(done)})}{m} = \frac{\sum_{i=1}^{m/2} (\frac{2}{m} + \frac{\epsilon}{m/2-1})}{m} = \frac{1+\epsilon}{m}$$

$$\text{UTILIZATION}_B = \frac{\sum_{i \in A \cup B} W_i^{(done)}}{m} = \frac{\epsilon}{m}$$

... *Almost no work is done!*

Competitive ratio example – Cat. B

I/Os of Category B applications are completed



$$\text{EFFICIENCY}_B = \frac{\sum_{i \in A \cup B} (V_i^{(transferred)} + W_i^{(done)})}{m} = \frac{\sum_{i=1}^{m/2} (\frac{2}{m} + \frac{\epsilon}{m/2-1})}{m} = \frac{1+\epsilon}{m}$$

$$\text{UTILIZATION}_B = \frac{\sum_{i \in A \cup B} W_i^{(done)}}{m} = \frac{\epsilon}{m}$$

... *Almost no work is done!*

Competitive ratio example – Conclusion

$$\text{OBJ}(\mathcal{S},\mathcal{I}) \times \rho_{\text{lb}} < \text{OBJ}(\text{OPTIMAL},\mathcal{I})$$

$$\text{EFFICIENCY}_B \times \frac{m}{4} = \frac{1+\epsilon}{m} \times \frac{m}{4} < \frac{m+2}{4m} = \text{EFFICIENCY}_A < \text{OPT}$$

$$\text{UTILIZATION}_B \times \frac{m}{4\epsilon} = \frac{\epsilon}{m} \times \frac{m}{4\epsilon} < \frac{m-2}{4m} = \text{UTILIZATION}_A < \text{OPT}$$

$$\text{MINYIELD}_B = \text{MINYIELD}_A = 0 \times \infty < \text{OPT}$$

(strictly positive yield obtained by sharing the bandwidth between all applications, so that they can all progress)

Competitive ratio example – Conclusion

$$\textsc{Obj}(\mathcal{S},\mathcal{I}) \times \rho_{\mathbf{lb}} < \textsc{Obj}(\textsc{Optimal},\mathcal{I})$$

$$\textsc{Efficiency}_B \times \frac{\mathbf{m}}{\mathbf{4}} = \frac{1+\epsilon}{m} \times \frac{\mathbf{m}}{\mathbf{4}} \quad < \quad \frac{m+2}{4m} = \textsc{Efficiency}_A < \textsc{Opt}$$

$$\textsc{Utilization}_B \times \frac{\mathbf{m}}{\mathbf{4\epsilon}} = \frac{\epsilon}{m} \times \frac{\mathbf{m}}{\mathbf{4\epsilon}} \quad < \quad \frac{m-2}{4m} = \textsc{Utilization}_A < \textsc{Opt}$$

$$\textsc{MinYield}_B = \textsc{MinYield}_A = 0 \times \infty < \textsc{Opt}$$

(strictly positive yield obtained by sharing the bandwidth between all applications, so that they can all progress)

Competitive ratio example – Conclusion

$$\textsc{Obj}(\mathcal{S},\mathcal{I}) \times \rho_{\mathbf{lb}} < \textsc{Obj}(\textsc{Optimal},\mathcal{I})$$

$$\textsc{Efficiency}_B \times \frac{\mathbf{m}}{\mathbf{4}} = \frac{1+\epsilon}{m} \times \frac{\mathbf{m}}{\mathbf{4}} \quad < \quad \frac{m+2}{4m} = \textsc{Efficiency}_A < \textsc{Opt}$$

$$\textsc{Utilization}_B \times \frac{\mathbf{m}}{\mathbf{4}\epsilon} = \frac{\epsilon}{m} \times \frac{\mathbf{m}}{\mathbf{4}\epsilon} \quad < \quad \frac{m-2}{4m} = \textsc{Utilization}_A < \textsc{Opt}$$

$$\textsc{MinYield}_B = \textsc{MinYield}_A = 0 \times \infty < \textsc{Opt}$$

(strictly positive yield obtained by sharing the bandwidth between all applications, so that they can all progress)

Competitive ratio example – Conclusion

$$\textsc{Obj}(\mathcal{S},\mathcal{I}) \times \rho_{\mathbf{lb}} < \textsc{Obj}(\textsc{Optimal},\mathcal{I})$$

$$\textsc{Efficiency}_B \times \frac{\mathbf{m}}{\mathbf{4}} = \frac{1+\epsilon}{m} \times \frac{\mathbf{m}}{\mathbf{4}} \quad < \quad \frac{m+2}{4m} = \textsc{Efficiency}_A < \textsc{Opt}$$

$$\textsc{Utilization}_B \times \frac{\mathbf{m}}{\mathbf{4}\epsilon} = \frac{\epsilon}{m} \times \frac{\mathbf{m}}{\mathbf{4}\epsilon} \quad < \quad \frac{m-2}{4m} = \textsc{Utilization}_A < \textsc{Opt}$$

$$\textsc{MinYield}_B = \textsc{MinYield}_A = 0 \times \infty < \textsc{Opt}$$

(strictly positive yield obtained by sharing the bandwidth between all applications, so that they can all progress)

Lower bounds on competitive ratios

	MinYield	Efficiency	Utilization
FairShare	m $\sqrt{m} - 3$ without history	$\frac{m}{4}$	∞
FCFS	∞	m	∞
Set-10	∞	m	∞
GreedyYield	∞	m	∞
GreedyCom	∞	$\frac{m}{4}$	∞
LookAheadGreedyYield	∞	m	∞
PeriodicGreedyYield ($\delta \to 0$)	2	m	∞
BestNextEv	$\frac{m}{2} - 4$	m	∞
Any strategy	$\frac{3}{2}$	$\frac{m}{4}$	∞

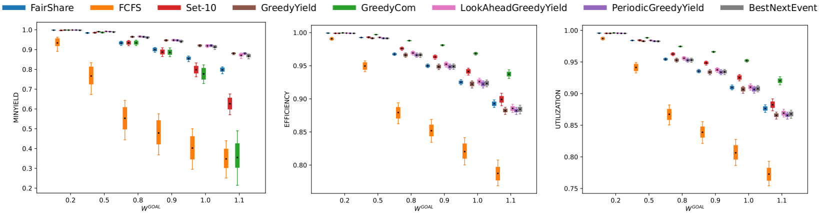
Outline

I/O pressure and synthetic traces

- Window $[T_{begin}, T_{end}]$, with m applications
- V_i: Volume that $\mathcal{A}_i$ would be able to transfer if it was executed in dedicated mode throughout the window; $V = \sum_{i=1}^{m} V_i$
- I/O pressure: $W = \frac{V}{B(T_{end} - T_{begin})}$

- Synthetic traces: Follow the methodology of [Boito et al. 2022]: $m = 60$ applications; $T_{end} - T_{begin} = 2\ 000\ 000$
- For each application $\mathcal{A}_i$:
  - Randomly generate average iteration length (normal distrib.)
  - Time spent on I/O: random parameter u_i uniformly picked in $[0, 1]$
  - Fraction of I/O: $\phi_i = \frac{u_i W^{GOAL}}{\sum_{k=1}^{m} u_k}$
  - Noise parameter ν_i to generate iterations of different lengths

Impact of I/O pressure



- New greedy strategies (except GREEDYCOM) very good for MINYIELD, much better than state-of-the-art competitors FCFS, FAIRSHARE and SET-10.
- EFFICIENCY and UTILIZATION: GREEDYCOM is the best (favors short communications)
- Complex strategy BESTNEXTEV not superior to simpler strategies

APEX workloads – http://www.nersc.gov/assets/apex-workflows-v2.pdf

Two very different workloads:

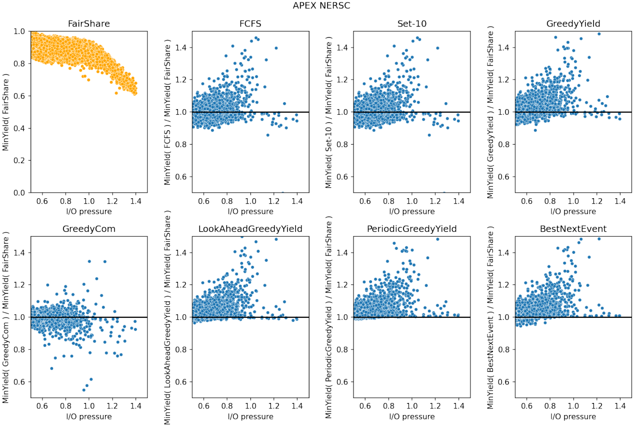
- NERSC – Large number of small apps (e.g., 24 cores for 4 hours); Some large apps (e.g., 16,512 cores, or 1/8 of the platform, for 48 hours); Some very long running apps (e.g., 10 days over 8,000 cores)
- TRILAB – More homogeneous set of apps (4096 to 32768 cores); Run for a significantly longer time (from 64h for the smallest duration, and up to 12 days)
- Application I/Os – All inputs read at the beginning; Checkpoints performed every hour; All outputs written at the end

Celio system:

- Workloads represent small I/O pressure (about 0.15 in average)
- Ratio between PFS bandwidth and computing performance of HPC platforms has a clearly decreasing trend ⇒ scaled versions of Celio

NERSC: MinYield of all strategies

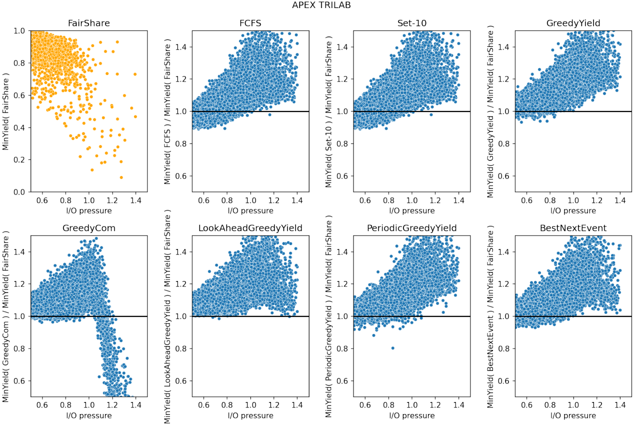
Ratio of the MinYield with the FairShare strategy



- LookAheadGY, PeriodicGY, and BestNextEv: very high probability of increasing MinYield compared to FairShare
- Higher performance increase with higher I/O pressure

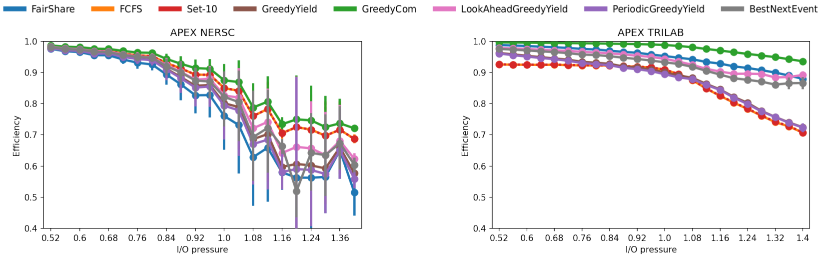
TRILAB: MinYield of all strategies

Ratio of the MinYield with the FairShare strategy



- Better than with NERSC, in particular GreedyCom: no performance drop, except with pressure > 1; LookAheadGY very good
- Again, higher I/O pressure $\Rightarrow$ need for efficient strategies

EFFICIENCY of all strategies for NERSC and TRILAB



- In terms of EFFICIENCY, GREEDYCOM is again the most efficient (but at the price of a lower MINYIELD)

- See [Benoit, Herault, Perotin, Robert, Vivien. *Revisiting I/O bandwidth-sharing strategies for HPC applications.* Inria RR-9502, 2023. Submitted; `https://inria.hal.science/hal-04038011v2`]

Outline

Conclusion – Take-aways

- **Current and future HPC platforms** demand simultaneous resource scheduling and resilience strategies for parallel applications
- Young/Daly formula commonly used to determine the optimal checkpointing period, but it is not always the best strategy **in practice** (periodic checkpointing might not be good!)
- Checkpoints $\Rightarrow$ I/O contention; Importance of bandwidth-sharing strategies, and first (lower bounds on) competitive ratios on the theoretical side
- **In practice**, LookAheadGreedyYield achieves excellent min yield on all scenarios; it achieves better utilization and efficiency than FairShare for NERSC and synthetic workloads, and the same performance for TRILAB;
  GreedyCom achieves the best performance for utilization and efficiency overall, but achieves poor min yield

Conclusion – Impact of failures

- **High-performance computers:** grow bigger and bigger, as Exaflop/s have been reached in June 2022 by Frontier (ORNL) – More than 8 millions cores, and obtains 52.23 gigaflops/watt
- High performance obtained at the price of huge energy consumption, even with *power-efficient systems*
- **Failures:** Redundant work and hence even larger energy consumption
- 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?

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

Future work

- Need for robust and resilient scheduling techniques for large-scale computing platforms ⇒ Two main axes for my future researches:
  1. Designing robust multi-criteria optimization algorithms (performance, reliability, energy), focusing in particular on edge-cloud platforms, when there are uncertainties about application properties but also on power sources (*variable capacity resources*; on-going project CNRS – U. Chicago)
  2. Designing new resilience techniques for Exascale, combining checkpoint with replication, and understanding how to efficiently select the resources to be used (PEPR NumPEx)
- Still a lot of algorithmic challenges to address, and techniques to be developed for many kinds of high-performance applications – both theoretical results and practical ones are expected ☺

My vision on the future of HPC

- Heading towards Zetta-scale? Rather than even bigger supercomputers, use of cluster collections, and distribution of computations; workflow migration, growing impact of I/Os
- Seems mandatory to *play* with flexibility (position paper following workshop with academics/industrials)
  - Flexible power in data centers (machines at risk, decide which jobs to kill/migrate)
  - Flexible workloads (Google: mandatory application part, but also optional, more flexible part)
- Care about energy consumption
  - Handle failures the best possible way
  - Beware of the rebound effect and encourage sobriety

Thanks!