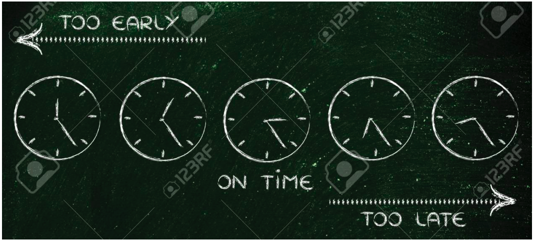
Performance at Scale: Scheduling Matters



Yves Robert
Ecole Normale Supérieure de Lyon, France
University of Tennessee Knoxville, USA

HPCS – July 17, 2019
http://graal.ens-lyon.fr/~yrobert/hpcs-dublin.pdf

Agenda: some scheduling problems

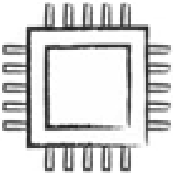
1. Scheduling checkpoints

2. Scheduling against IO interference

3. Scheduling for replication

4. Scheduling stochastic tasks

Scale is the enemy

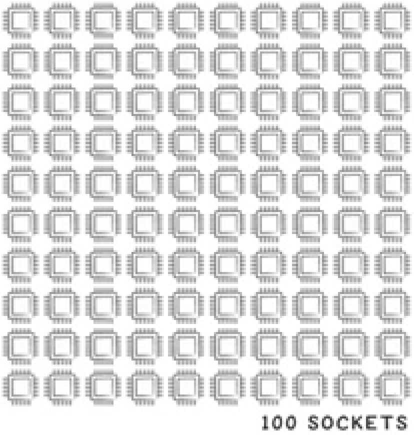


100

YEARS

MEAN TIME
BETWEEN FAILURES

Scale is the enemy

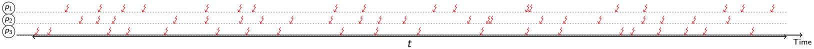


1

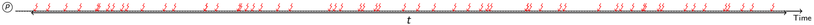
YEAR

MEAN TIME BETWEEN FAILURES

Scale is the enemy



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

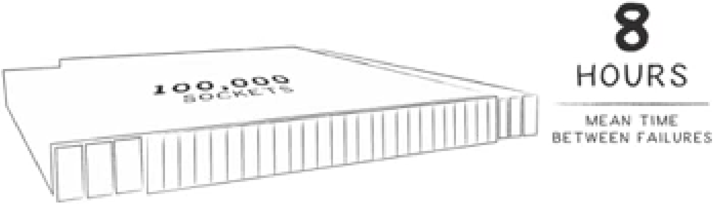


...during the same time, the platform has around 60 faults ($\mu_p = \frac{t}{60}$)

Scale is the enemy

1,000
SOCKETS
36
DAYS
MEAN TIME
BETWEEN FAILURES

Scale is the enemy

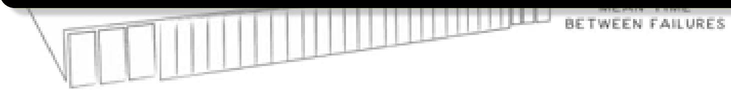


Scale is the enemy

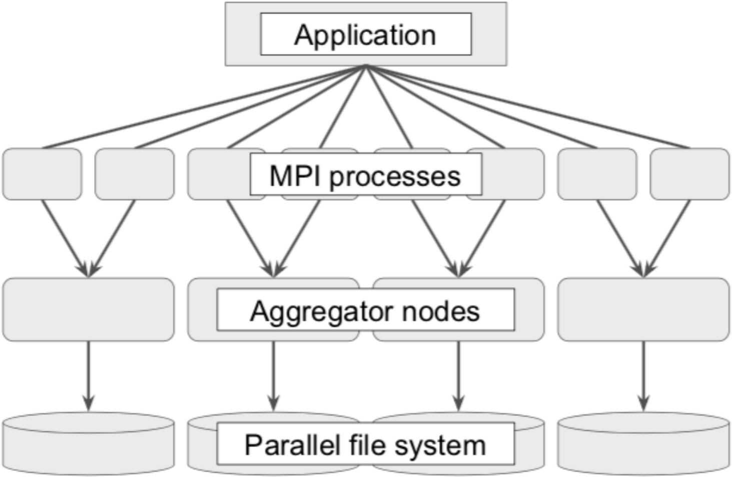
Need to checkpoint!

But when?

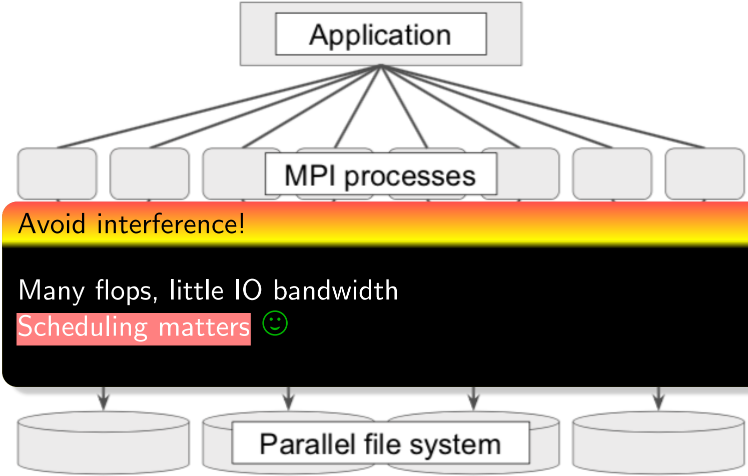
Scheduling matters ☺



IO gap increases



IO gap increases

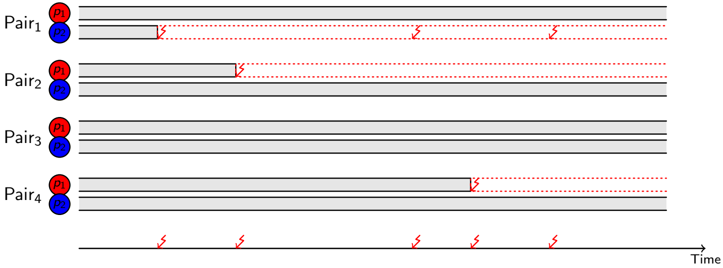


Avoid interference!

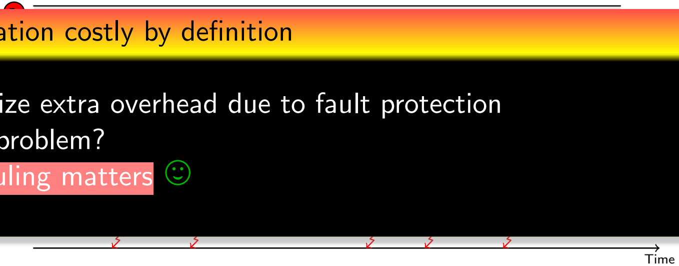
Many flops, little IO bandwidth

Scheduling matters ☺

Replication



Replication



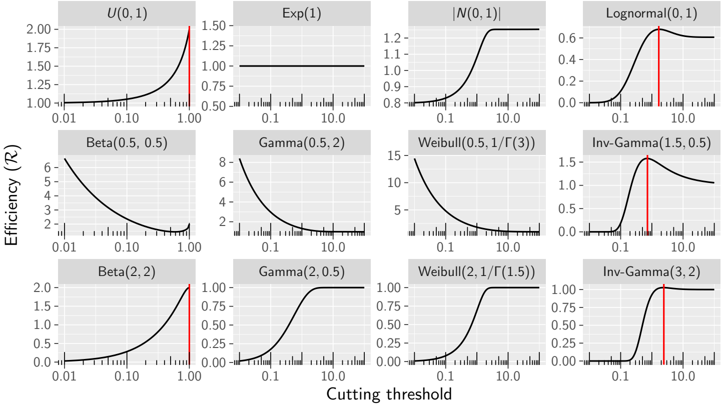
Replication costly by definition

Minimize extra overhead due to fault protection

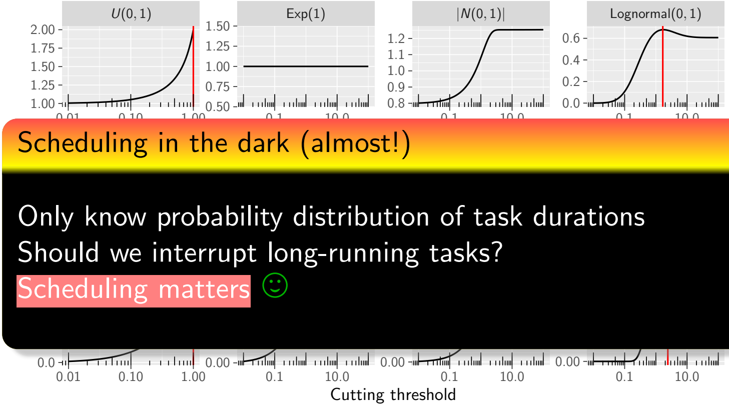
Open problem?

Scheduling matters ☺

Stochastic tasks



Stochastic tasks



Scheduling in the dark (almost!)

Only know probability distribution of task durations

Should we interrupt long-running tasks?

Scheduling matters ☺

Outline

1. Scheduling checkpoints
2. IO Contention
3. Replication for fail-stop failures
4. Scheduling Stochastic Tasks
5. Conclusion

Outline

1. Scheduling checkpoints
   - Faults and failures
   - Checkpointing
   - In-memory checkpointing
   - Multi-level checkpointing
   - Silent errors
2. IO Contention
3. Replication for fail-stop failures
4. Scheduling Stochastic Tasks
5. Conclusion

Outline

1 Scheduling checkpoints
- Faults and failures
- Checkpointing
- In-memory checkpointing
- Multi-level checkpointing
- Silent errors

2 IO Contention

3 Replication for fail-stop failures

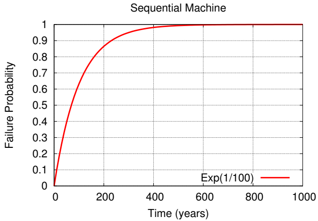
4 Scheduling Stochastic Tasks

5 Conclusion

Definitions

- Many types of faults: software error, hardware malfunction, memory corruption
- Many possible behaviors: silent, transient, unrecoverable
- Restrict to faults that lead to application failures
- This includes all hardware faults, and some software ones
- Will use terms *fault* and *failure* interchangeably
- Silent errors (Silent Data Corruptions) addressed later

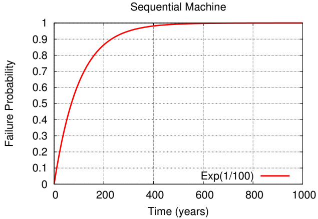
Failure distributions: (1) Exponential



Exp(λ): Exponential distribution law of parameter λ:

- Pdf: $f(t) = \lambda e^{-\lambda t} dt$ for $t \geq 0$
- Cdf: $F(t) = 1 - e^{-\lambda t}$
- Mean $= \frac{1}{\lambda}$

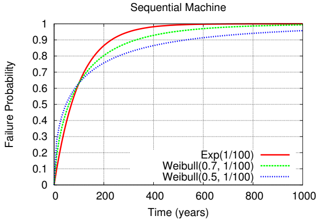
Failure distributions: (1) Exponential



X random variable for $Exp(\lambda)$ failure inter-arrival times:

- $\mathbb{P}(X \le t) = 1 - e^{-\lambda t} dt$ (by definition)
- Memoryless property: $\mathbb{P}\left(X \ge t + s \,|\, X \ge s\right) = \mathbb{P}(X \ge t)$ at any instant, time to next failure does not depend upon time elapsed since last failure
- Mean Time Between Failures (MTBF) $\mu = \mathbb{E}\left(X\right) = \frac{1}{\lambda}$

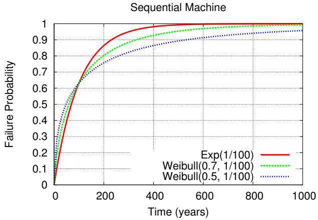
Failure distributions: (2) Weibull



Weibull(k,λ): 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 $= \frac{1}{\lambda}\Gamma(1+\frac{1}{k})$

Failure distributions: (2) Weibull



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

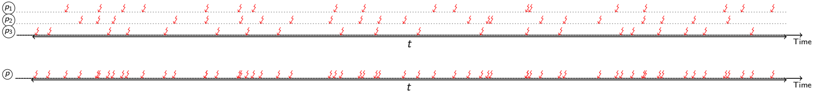
- If $k < 1$: failure rate decreases with time
  "infant mortality": defective items fail early
- If $k = 1$: $\mathit{Weibull}(1, \lambda) = \mathit{Exp}(\lambda)$ constant failure time

Platform MTBF

- Rebooting only faulty processor
- Platform failure distribution
  ⇒ superposition of p IID processor distributions of MTBF μ
  ⇒ 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

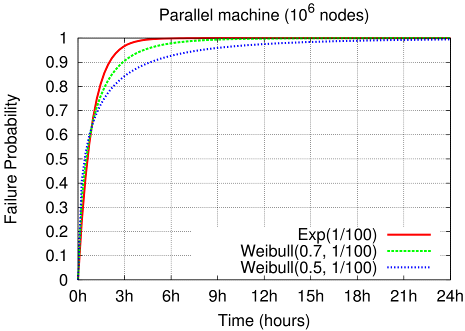


Theorem: $\mu_p = \dfrac{\mu}{p}$ for arbitrary distributions

Values from the literature

- MTBF of one processor: between 1 and 125 years
- Shape parameters for Weibull: $k = 0.5$ or $k = 0.7$
- Failure trace archive from INRIA (`http://fta.inria.fr`)
- Computer Failure Data Repository from LANL (`http://institutes.lanl.gov/data/fdata`)

Does it matter?



After infant mortality and before aging,
instantaneous failure rate of computer platforms is almost constant

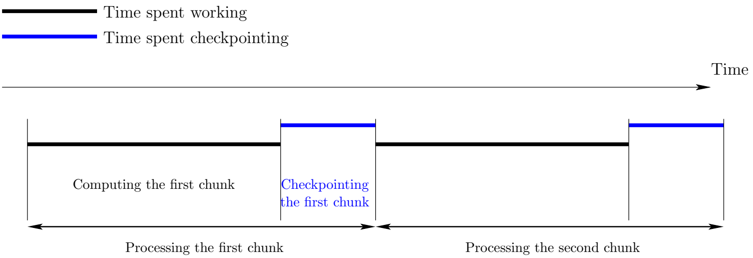
Summary for the road

- MTBF key parameter and $\mu_p = \frac{\mu}{p}$ ☺
- Exponential distribution OK for most purposes ☺
- Assume failure independence while not (completely) true ☹

Outline

1. Scheduling checkpoints
   - Faults and failures
   - Checkpointing
   - In-memory checkpointing
   - Multi-level checkpointing
   - Silent errors
2. IO Contention
3. Replication for fail-stop failures
4. Scheduling Stochastic Tasks
5. Conclusion

Periodic checkpointing


Time spent working
Time spent checkpointing
Time
Computing the first chunk
Checkpointing the first chunk
Processing the first chunk
Processing the second chunk

Blocking model: while a checkpoint is taken, no computation can be performed

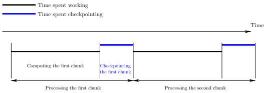
Framework

- Periodic checkpointing policy of period $T = W + C$
- Independent and identically distributed failures
- Applies to a single processor with MTBF $\mu = \mu_{ind}$
- Applies to a platform with p processors and MTBF $\mu = \frac{\mu_{ind}}{p}$
    - coordinated checkpointing
    - tightly-coupled application
    - progress $\Leftrightarrow$ all processors available

$\Rightarrow$ platform = single (powerful, unreliable) processor ☺

Waste: fraction of time not spent for useful computations

Waste in fault-free execution


Time spent working
Time spent checkpointing
Time
Computing the first chunk
Checkpointing the first chunk
Processing the first chunk
Processing the second chunk

- $\text{TIME}_{\text{base}}$: application base time
- TIME_{FF}: with periodic checkpoints but failure-free

$$\text{TIME}_{\text{FF}} = \text{TIME}_{\text{base}} + \#\mathit{checkpoints} \times C$$

$$\#\mathit{checkpoints} = \left\lceil \frac{\text{TIME}_{\text{base}}}{T - C} \right\rceil \approx \frac{\text{TIME}_{\text{base}}}{T - C} \text{ (valid for large jobs)}$$

$$\text{WASTE}[FF] = \frac{\text{TIME}_{\text{FF}} - \text{TIME}_{\text{base}}}{\text{TIME}_{\text{FF}}} = \frac{C}{T}$$

Waste due to failures

- $\textsc{Time}_{\text{base}}$: application base time
- $\textsc{Time}_{\text{FF}}$: with periodic checkpoints but failure-free
- $\textsc{Time}_{\text{final}}$: expectation of time with failures

$$\textsc{Time}_{\text{final}} = \textsc{Time}_{\text{FF}} + N_{faults} \times T_{\text{lost}}$$

N_{faults} number of failures during execution
T_{lost}: average time lost per failure

$$N_{faults} = \frac{\textsc{Time}_{\text{final}}}{\mu}$$

T_{lost}?

Waste due to failures

- $\textsc{Time}_{\text{base}}$: application base time
- $\textsc{Time}_{\text{FF}}$: with periodic checkpoints but failure-free
- $\textsc{Time}_{\text{final}}$: expectation of time with failures

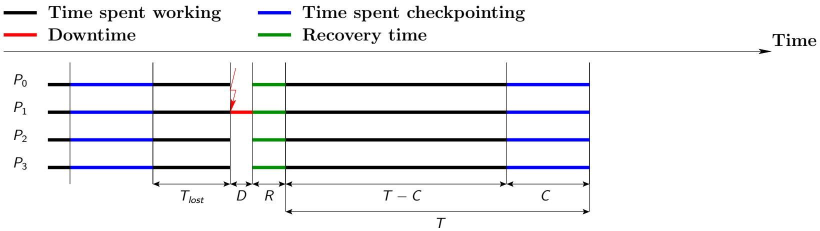
$$\textsc{Time}_{\text{final}} = \textsc{Time}_{\text{FF}} + N_{faults} \times T_{\text{lost}}$$

N_{faults} number of failures during execution
T_{lost}: average time lost per failure

$$N_{faults} = \frac{\textsc{Time}_{\text{final}}}{\mu}$$

T_{lost}?

Computing T_{lost}



$$T_{\text{lost}} = D + R + \frac{T}{2}$$

Rationale

⇒ Instants when periods begin and failures strike are independent

⇒ Approximation used for all distribution laws

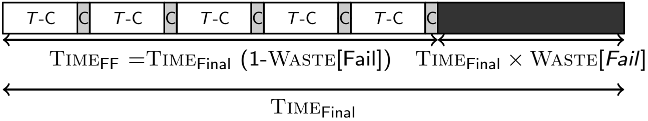
⇒ Exact for Exponential and uniform distributions

Waste due to failures

$$\textsc{Time}_{\mathsf{final}} = \textsc{Time}_{\mathsf{FF}} + N_{faults} \times T_{\mathsf{lost}}$$

$$\textsc{Waste}[fail] = \frac{\textsc{Time}_{\mathsf{final}} - \textsc{Time}_{\mathsf{FF}}}{\textsc{Time}_{\mathsf{final}}} = \frac{1}{\mu}\left(D + R + \frac{T}{2}\right)$$

Total waste

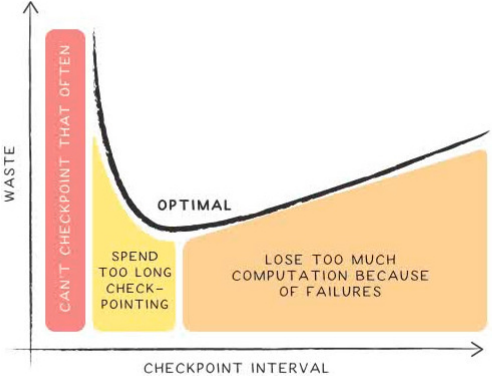

T-C
C
T-C
C
T-C
C
T-C
C
T-C
C
$\textsc{Time}_{\textsf{FF}} = \textsc{Time}_{\textsf{Final}}$ (1-$\textsc{Waste}$[Fail])
$\textsc{Time}_{\textsf{Final}} \times \textsc{Waste}[\mathit{Fail}]$
$\textsc{Time}_{\textsf{Final}}$

$$\textsc{Waste} = \frac{\textsc{Time}_{\textsf{final}} - \textsc{Time}_{\textsf{base}}}{\textsc{Time}_{\textsf{final}}}$$

$$1 - \textsc{Waste} = (1 - \textsc{Waste}[FF])(1 - \textsc{Waste}[fail])$$

$$\textsc{Waste} = \frac{C}{T} + \left(1 - \frac{C}{T}\right)\frac{1}{\mu}\left(D + R + \frac{T}{2}\right)$$

Optimal checkpointing interval



Waste minimization

$$\textsc{Waste} = \frac{C}{T} + \left(1 - \frac{C}{T}\right)\frac{1}{\mu}\left(D + R + \frac{T}{2}\right)$$

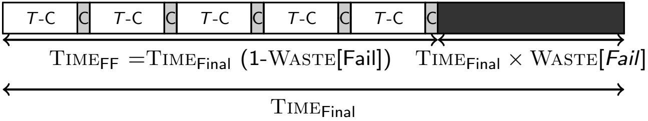
$$\textsc{Waste} = \frac{u}{T} + v + wT$$

$$u = C\left(1 - \frac{D+R}{\mu}\right) \qquad v = \frac{D+R-C/2}{\mu} \qquad w = \frac{1}{2\mu}$$

$\textsc{Waste}$ minimized for $T = \sqrt{\frac{u}{w}}$

$$T = \sqrt{2(\mu - (D+R))C}$$

Comparison with Young/Daly


T-C
C
T-C
C
T-C
C
T-C
C
T-C
C
$\textsc{Time}_{\mathsf{FF}} = \textsc{Time}_{\mathsf{Final}}$ (1-$\textsc{Waste}$[Fail])
$\textsc{Time}_{\mathsf{Final}} \times \textsc{Waste}[\mathit{Fail}]$
$\textsc{Time}_{\mathsf{Final}}$

$$\left(1 - \textsc{Waste}[\mathit{fail}]\right)\textsc{Time}_{\mathsf{final}} = \textsc{Time}_{\mathsf{FF}}$$
$$\Rightarrow T = \sqrt{2(\mu - (D+R))C}$$

Daly: $\textsc{Time}_{\mathsf{final}} = \left(1 + \textsc{Waste}[\mathit{fail}]\right)\textsc{Time}_{\mathsf{FF}}$
$$\Rightarrow T = \sqrt{2(\mu + (D+R))C} + C$$

Young: $\textsc{Time}_{\mathsf{final}} = \left(1 + \textsc{Waste}[\mathit{fail}]\right)\textsc{Time}_{\mathsf{FF}}$ and $D = R = 0$
$$\Rightarrow T = \sqrt{2\mu C} + C$$

Wrap up

- Capping periods, and enforcing a lower bound on MTBF
  ⇒ mandatory for mathematical rigor ☹

- Not needed for practical purposes ☺
  - actual job execution uses optimal value
  - account for multiple faults by re-executing work until success

- Approach surprisingly robust ☺

Lesson learnt for fail-stop failures

(Not so) Secret data

- Tsubame 2: 962 failures during last 18 months so $\mu = 13$ hrs
- Blue Waters: 2-3 node failures per day
- Titan: a few failures per day
- Tianhe 2: wouldn't say

$$T_{\text{opt}} = \sqrt{2\mu C} \quad \Rightarrow \quad \textsc{Waste}_{opt} \approx \sqrt{\frac{2C}{\mu}}$$

Petascale:	$C = 20$ min	$\mu = 24$ hrs	$\Rightarrow \textsc{Waste}_{opt} = 17\%$
Scale by 10:	$C = 20$ min	$\mu = 2.4$ hrs	$\Rightarrow \textsc{Waste}_{opt} = 53\%$
Scale by 100:	$C = 20$ min	$\mu = 0.24$ hrs	$\Rightarrow \textsc{Waste}_{opt} = 100\%$

Lesson learnt for fail-stop failures

(so) Secret data

- Tsuban 962 failures during last 18 months so 13 hrs
- Blue Waters: 2- de failures per day
- Titan: a few failures pe y
- Tianhe

Exascale $\neq$ Petascale $\times 1000$

Need more reliable components

Need to checkpoint faster

Petascale	$C = 20$ min	$\mu = 24$ hrs	$\Rightarrow$ W TE$_{opt} = 17\%$
Scale 10:	$C = 20$ min	$\mu = 2.4$ hrs	$\Rightarrow$ WAS $_{opt} = 53\%$
Sca by 100:	$C = 20$ min	$\mu = 0.24$ hrs	$\Rightarrow$ WASTE $= 100\%$

Lesson learnt for fail-stop failures

(Not so) Secret data

- Tsubame 2: 962 failures during last 18 months so $\mu = 13$ hrs
- Blue Waters: 2-3 node failures per day
- Titan: a few failures per day
- Tianhe 2: wouldn't say

Silent errors:

detection latency ⇒ additional problems

Petascale:	$C = 20$ min	$\mu = 24$ hrs	$\Rightarrow \textsc{Waste}_{opt} = 17\%$
Scale by 10:	$C = 20$ min	$\mu = 2.4$ hrs	$\Rightarrow \textsc{Waste}_{opt} = 53\%$
Scale by 100:	$C = 20$ min	$\mu = 0.24$ hrs	$\Rightarrow \textsc{Waste}_{opt} = 100\%$

Exponential distributions

Compute the expected time $\mathbb{E}(W)$ to execute a work of duration W followed by a checkpoint of duration C.

Recursive Approach

$$\mathbb{E}(W) =$$

Exponential distributions

Compute the expected time $\mathbb{E}(W)$ to execute a work of duration W followed by a checkpoint of duration C.

Recursive Approach

$$\mathbb{E}(W) = \overbrace{\mathcal{P}_{\text{succ}}(W+C)}^{\substack{\text{Probability}\\ \text{of success}}}(W+C)$$

Exponential distributions

Compute the expected time $\mathbb{E}(W)$ to execute a work of duration W followed by a checkpoint of duration C.

Recursive Approach

$$\mathbb{E}(W) = \mathcal{P}_{\text{succ}}(W+C)\,\overbrace{(W+C)}^{\substack{\text{Time needed}\\ \text{to compute}\\ \text{the work } W \text{ and}\\ \text{checkpoint it}}}$$

Exponential distributions

Compute the expected time $\mathbb{E}(W)$ to execute a work of duration W followed by a checkpoint of duration C.

Recursive Approach

$$\mathbb{E}(W) = \begin{array}{l} \mathcal{P}_{\text{succ}}(W+C)\,(W+C) \\ + \\ \underbrace{(1-\mathcal{P}_{\text{succ}}(W+C))}_{\text{Probability of failure}}\,(\mathbb{E}(T_{lost}(W+C)) + \mathbb{E}(T_{rec}) + \mathbb{E}(W)) \end{array}$$

Exponential distributions

Compute the expected time $\mathbb{E}(W)$ to execute a work of duration W followed by a checkpoint of duration C.

Recursive Approach

$$\mathbb{E}(W) = \begin{array}{l} \mathcal{P}_{\text{succ}}(W+C)\,(W+C) \\ + \\ (1-\mathcal{P}_{\text{succ}}(W+C))\,(\underbrace{\mathbb{E}(T_{lost}(W+C))}_{\substack{\text{Time elapsed}\\ \text{before failure}\\ \text{stroke}}} + \mathbb{E}(T_{rec}) + \mathbb{E}(W)) \end{array}$$

Exponential distributions

Compute the expected time $\mathbb{E}(W)$ to execute a work of duration W followed by a checkpoint of duration C.

Recursive Approach

$$\mathbb{E}((W) = \begin{array}{l} \mathcal{P}_{\text{succ}}(W+C)\,(W+C) \\ + \\ (1-\mathcal{P}_{\text{succ}}(W+C))\,(\mathbb{E}(T_{lost}(W+C)) + \underbrace{\mathbb{E}(T_{rec})}_{\substack{\text{Time needed}\\\text{to perform}\\\text{downtime}\\\text{and recovery}}} + \mathbb{E}(W)) \end{array}$$

Exponential distributions

Compute the expected time $\mathbb{E}(W)$ to execute a work of duration W followed by a checkpoint of duration C.

Recursive Approach

$$\mathbb{E}(W) = \begin{array}{l} \mathcal{P}_{\text{succ}}(W+C)\,(W+C) \\ + \\ (1-\mathcal{P}_{\text{succ}}(W+C))\,(\mathbb{E}(T_{lost}(W+C)) + \mathbb{E}(T_{rec}) + \underbrace{\mathbb{E}(W)}_{\substack{\text{Time needed} \\ \text{to compute } W \\ \text{anew}}}) \end{array}$$

Computation of $\mathbb{E}(W)$

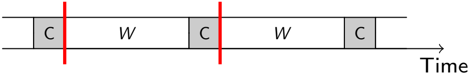
$$\mathbb{E}(W) = \begin{array}{l} \mathcal{P}_{\text{succ}}(W+C)\,(W+C) \\ + \\ (1-\mathcal{P}_{\text{succ}}(W+C))\,(\mathbb{E}(T_{lost}(W+C)) + \mathbb{E}(T_{rec}) + \mathbb{E}(W)) \end{array}$$

- $\mathbb{P}_{suc}(W+C) = e^{-\lambda(W+C)}$
- $\mathbb{E}(T_{lost}(W+C)) = \int_0^\infty x\mathbb{P}(X = x | X < W+C)dx = \frac{1}{\lambda} - \frac{W+C}{e^{\lambda(W+C)}-1}$
- $\mathbb{E}(T_{rec}) = e^{-\lambda R}(D+R)+(1-e^{-\lambda R})(D+\mathbb{E}(T_{lost}(R))+\mathbb{E}(T_{rec}))$

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

Optimal checkpointing interval

Minimize expected execution overhead $H(W) = \frac{\mathbb{E}(W)}{W} - 1$



- Exact solution:

$$H(W) = \frac{e^{\lambda R}(\frac{1}{\lambda} + D)e^{\lambda(W+C)}}{W} - 1, \text{use Lambert function}$$

- First-order approximation [Young/Daly]:

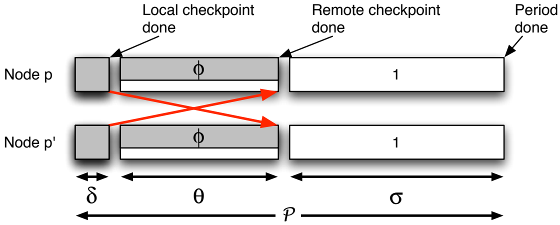
$$W_{\text{opt}} = \sqrt{\frac{2C}{\lambda}} = \sqrt{2C\mu}$$

$$H_{\text{opt}} = \sqrt{2\lambda C} + \Theta(\lambda)$$

Outline

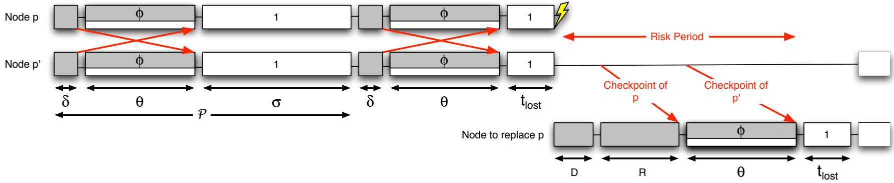
1. Scheduling checkpoints
   - Faults and failures
   - Checkpointing
   - In-memory checkpointing
   - Multi-level checkpointing
   - Silent errors
2. IO Contention
3. Replication for fail-stop failures
4. Scheduling Stochastic Tasks
5. Conclusion

Double checkpoint algorithm (Kale et al., UIUC)



- Platform nodes partitioned into pairs
- Each node in a pair exchanges its checkpoint with its *buddy*
- Each node saves two checkpoints:
  - one locally: storing its own data
  - one remotely: receiving and storing its buddy's data

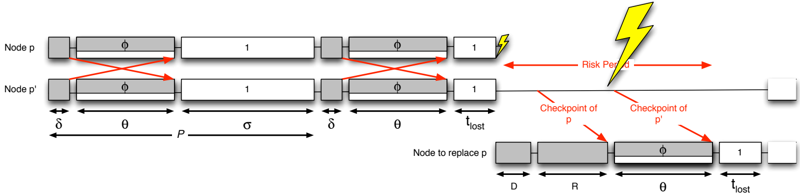
Failures



- After failure: downtime D and recovery from buddy node
- Two checkpoint files lost, must be re-sent to faulty processor

Best trade-off between performance and risk?

Failures



- After failure: downtime D and recovery from buddy node
- Two checkpoint files lost, must be re-sent to faulty processor
- Application at risk until complete reception of both messages

Best trade-off between performance and risk?

Outline

1. Scheduling checkpoints
   - Faults and failures
   - Checkpointing
   - In-memory checkpointing
   - Multi-level checkpointing
   - Silent errors
2. IO Contention
3. Replication for fail-stop failures
4. Scheduling Stochastic Tasks
5. Conclusion

Multi-level checkpointing

Coordinated checkpointing
⇒ **Scalability problem for large-scale platforms**

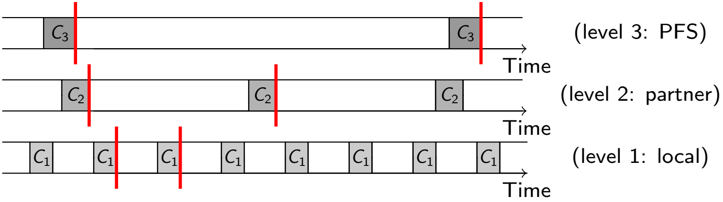
Multiple technologies to cope with different failure types:

- Local memory/SSD
- Partner copy/XOR
- Reed-Solomon coding
- Parallel file system

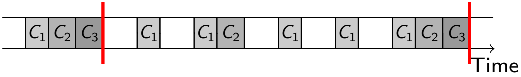
Scalable Checkpoint/Restart (SCR) library
Fault Tolerance Interface (FTI)
VeloC (ECP project)

Simplified model

- Independent checkpointing:

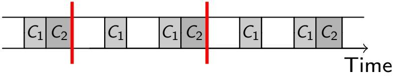


- Synchronized checkpointing:



Two Levels

Easier because pattern repeats (memoryless property)



- Exact solution: very complicated (which error type occurs first?), equal-length chunks, see [1]
- First-order approximation:

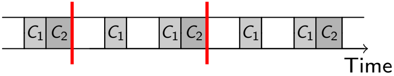
$$H_{\text{opt}} = \sqrt{2\lambda_1 C_1} + \sqrt{2\lambda_2 C_2} + \Theta(\lambda)$$

(obtained for some optimal pattern)

[1] S. Di, Y. Robert, F. Vivien, F. Cappello. Toward an optimal online checkpoint solution under a two-level HPC checkpoint model, *IEEE TPDS*, 2017.

Two Levels

Easier because pattern repeats (memoryless property)



- Exact solution: very complicated (which error type occurs first?), equal-length chunks, see [1]
- First-order approximation:

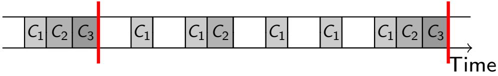
$$H_{\text{opt}} = \sqrt{2\lambda_1 C_1} + \sqrt{2\lambda_2 C_2} + \Theta(\lambda)$$

(obtained for some optimal pattern)

[1] S. Di, Y. Robert, F. Vivien, F. Cappello. Toward an optimal online checkpoint solution under a two-level HPC checkpoint model, *IEEE TPDS*, 2017.

Three Levels

Difficult because sub-patterns may differ



- Exact solution: unknown
- First-order approximation:

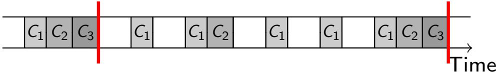
$$H_{\text{opt}} = \sqrt{2\lambda_1 C_1} + \sqrt{2\lambda_2 C_2} + \sqrt{2\lambda_3 C_3} + \Theta(\lambda)$$

- Choose optimal set of levels:

Level	Overhead
1, 2, 3	$\sqrt{2C_1\lambda_1} + \sqrt{2C_2\lambda_2} + \sqrt{2C_3\lambda_3}$
1, 3	$\sqrt{2C_1\lambda_1} + \sqrt{2C_3(\lambda_2+\lambda_3)}$
2, 3	$\sqrt{2C_2(\lambda_1+\lambda_2)} + \sqrt{2C_3\lambda_3}$
3	$\sqrt{2C_3(\lambda_1+\lambda_2+\lambda_3)}$

Three Levels

Difficult because sub-patterns may differ


C_1 C_2 C_3 C_1 C_1 C_2 C_1 C_1 C_1 C_2 C_3
Time

- Exact solution: unknown
- First-order approximation:

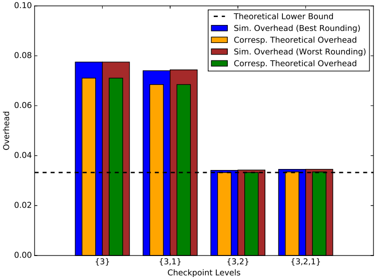
$$H_{\text{opt}} = \sqrt{2\lambda_1 C_1} + \sqrt{2\lambda_2 C_2} + \sqrt{2\lambda_3 C_3} + \Theta(\lambda)$$

- Choose optimal set of levels:

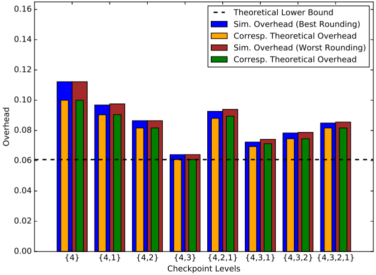
Level	Overhead
$1,2,3$	$\sqrt{2C_1\lambda_1} + \sqrt{2C_2\lambda_2} + \sqrt{2C_3\lambda_3}$
$1,3$	$\sqrt{2C_1\lambda_1} + \sqrt{2C_3(\lambda_2+\lambda_3)}$
$2,3$	$\sqrt{2C_2(\lambda_1+\lambda_2)} + \sqrt{2C_3\lambda_3}$
3	$\sqrt{2C_3(\lambda_1+\lambda_2+\lambda_3)}$

Simulations

Set	Source	Level	1	2	3	4
(A)	Moody et al. [1]	C (s)	0.5	4.5	1051	-
		MTBF (s)	5.00e6	5.56e5	2.50e6	-
(B)	Balaprakash et al. [2]	C (s)	10	20	20	100
		MTBF (s)	3.60e4	7.20e4	1.44e5	7.20e5



(A)



(B)

[1] A. Moody, G. Bronevetsky, K. Mohror, and B. R. de Supinski. Design, modeling, and evaluation of a scalable multi-level checkpointing system. *Supercomputing*, 2010.

[2] P. Balaprakash, L. A. Bautista-Gomez, M.-S. Bouguerra, S. M. Wild, F. Cappello, and P. D. Hovland. Analysis of the tradeoffs between energy and run time for multilevel checkpointing. *PMBS*, 2014.

Takeaway

Explicit formulas for (almost) optimal multi-level checkpointing

$$H_{\text{opt}} = \sum_{\ell=1}^{k} \sqrt{2\lambda_\ell C_\ell} + \Theta(\lambda)$$

Limitations:

- First-order accurate for platform MTBF in hours $\Longleftrightarrow$ 10,000s of nodes. Beyond?
- Independent errors ☹
  Correlated failures across levels?

[1] A. Benoit, A. Cavelan, Y. Robert and H. Sun. Towards optimal multi-level checkpointing, *IEEE TC*, 2017.

Outline

1. Scheduling checkpoints
   - Faults and failures
   - Checkpointing
   - In-memory checkpointing
   - Multi-level checkpointing
   - Silent errors
2. IO Contention
3. Replication for fail-stop failures
4. Scheduling Stochastic Tasks
5. Conclusion

Definitions

- Instantaneous error detection ⇒ fail-stop failures, e.g. resource crash
- Silent errors (data corruption) ⇒ detection latency

Silent error detected only when the corrupt data is activated

- Includes some software faults, some hardware errors (soft errors in L1 cache), double bit flip
- Cannot always be corrected by ECC memory

Probability distributions for silent errors

?

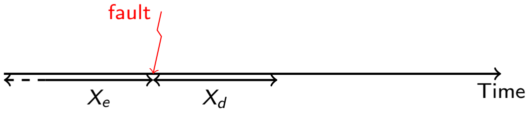
Theorem: $\mu_p = \frac{\mu_{\text{ind}}}{p}$ for arbitrary distributions

Probability distributions for silent errors

?

Theorem: $\mu_p = \frac{\mu_{\text{ind}}}{p}$ for arbitrary distributions

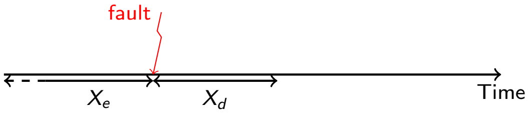
General-purpose approach



Error and detection latency

- Last checkpoint may have saved an already corrupted state
- Saving k checkpoints (Lu, Zheng and Chien):
  1. Critical failure when all live checkpoints are invalid
  2. Which checkpoint to roll back to?

General-purpose approach



Error and detection latency

- Last checkpoint may have saved an already corrupted state
- Saving *k* checkpoints (Lu, Zheng and Chien):
  1. Critical failure when all live checkpoints are invalid
     Assume unlimited storage resources
  2. Which checkpoint to roll back to?
     Assume verification mechanism

Limitation of the model

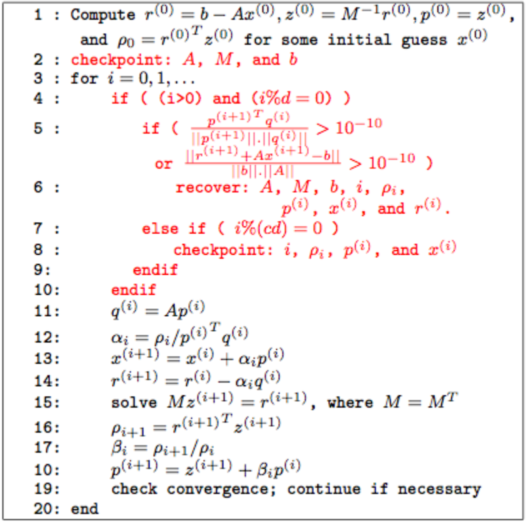
It is not clear how to detect when the error has occurred (hence to identify the last valid checkpoint) ☹ ☹ ☹

Need a verification mechanism to check the correctness of the checkpoints. This has an additional cost!

Coupling checkpointing and verification

- Verification mechanism of cost V
- Silent errors detected only when verification is executed
- Approach agnostic of the nature of verification mechanism (checksum, error correcting code, coherence tests, etc)
- Fully general-purpose
  (application-specific information, if available, can always be used to decrease V)

On-line ABFT scheme for PCG

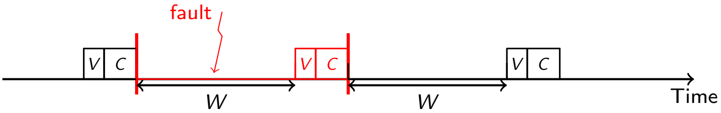


```
Compute $r^{(0)} = b - Ax^{(0)}, z^{(0)} = M^{-1}r^{(0)}, p^{(0)} = z^{(0)}$,
  and $\rho_0 = {r^{(0)}}^T z^{(0)}$ for some initial guess $x^{(0)}$
checkpoint: $A$, $M$, and $b$
for $i = 0, 1, \dots$
    if ( (i>0) and ($i\%d = 0$) )
        if ( $\frac{{p^{(i+1)}}^T q^{(i)}}{||p^{(i+1)}||.||q^{(i)}||} > 10^{-10}$
          or $\frac{||r^{(i+1)}+Ax^{(i+1)}-b||}{||b||.||A||} > 10^{-10}$ )
            recover: $A$, $M$, $b$, $i$, $\rho_i$,
                $p^{(i)}$, $x^{(i)}$, and $r^{(i)}$.
        else if ( $i\%(cd) = 0$ )
            checkpoint: $i$, $\rho_i$, $p^{(i)}$, and $x^{(i)}$
        endif
    endif
    $q^{(i)} = Ap^{(i)}$
    $\alpha_i = \rho_i / {p^{(i)}}^T q^{(i)}$
    $x^{(i+1)} = x^{(i)} + \alpha_i p^{(i)}$
    $r^{(i+1)} = r^{(i)} - \alpha_i q^{(i)}$
    solve $Mz^{(i+1)} = r^{(i+1)}$, where $M = M^T$
    $\rho_{i+1} = {r^{(i+1)}}^T z^{(i+1)}$
    $\beta_i = \rho_{i+1} / \rho_i$
    $p^{(i+1)} = z^{(i+1)} + \beta_i p^{(i)}$
    check convergence; continue if necessary
end
```

Zizhong Chen, PPoPP'13

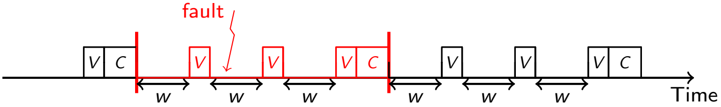
- Iterate PCG
  Cost: SpMV, preconditioner solve, 5 linear kernels
- Detect soft errors by checking orthogonality and residual
- Verification every *d* iterations
  Cost: scalar product+SpMV
- Checkpoint every *c* iterations
  Cost: three vectors, or two vectors + SpMV at recovery
- Experimental method to choose *c* and *d*

Base pattern (and revisiting Young/Daly)



	Fail-stop (classical)	Silent errors
Pattern	$T = W + C$	$T = W + V + C$
$\textsc{Waste}[FF]$	$\frac{C}{T}$	$\frac{V+C}{T}$
$\textsc{Waste}[fail]$	$\frac{1}{\mu}(D + R + \frac{T}{2})$	$\frac{1}{\mu}(R + T + V)$
Optimal	$T_{\text{opt}} = \sqrt{2C\mu}$	$T_{\text{opt}} = \sqrt{(V + C)\mu}$
$\textsc{Waste}_{opt}$	$\sqrt{\frac{2C}{\mu}}$	$2\sqrt{\frac{C+V}{\mu}}$

With $p = 1$ checkpoint and $q = 3$ verifications



Base Pattern	$p = 1, q = 1$	$\text{WASTE}_{opt} = 2\sqrt{\frac{C+V}{\mu}}$
New Pattern	$p = 1, q = 3$	$\text{WASTE}_{opt} = 2\sqrt{\frac{4(C+3V)}{6\mu}}$

Application-specific methods (1/2)

- ABFT: dense matrices / fail-stop, extended to sparse / silent. Limited to one error detection and/or correction in practice
- Asynchronous (chaotic) iterative methods (old work)
- Partial differential equations: use lower-order scheme as verification mechanism (detection only, Benson, Schmit and Schreiber)
- FT-GMRES: inner-outer iterations (Hoemmen and Heroux)
- PCG: orthogonalization check every *k* iterations, re-orthogonalization if problem detected (Sao and Vuduc)
- Algorithm-based focused recovery: use application data-flow to identify potential error source and corrupted nodes (Fang and Chien 2014)

Application-specific methods (2/2)

- Dynamic monitoring of datasets based on physical laws (e.g., temperature/speed limit) and space or temporal proximity (Bautista-Gomez and Cappello)
- Time-series prediction, spatial multivariate interpolation (Di et al.)
- Offline training, online detection based on SDC signature for convergent iterative applications (Liu and Agrawal)
- Spatial regression based on support vector machines (Subasi et al.)
- Many others data-analytics/machine learning approaches

Application-specific detectors

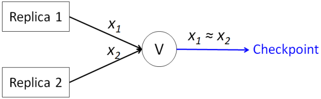
Do you believe it?

- Detectors are not perfect
- High recall is expensive if at all achievable
- With higher error rates, it would be good to correct a few errors

Replication mandatory at scale? ☹

Why Is Replication Useful?

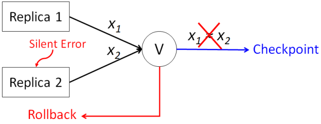
- **Error detection (duplication)**:



- **Error correction (triplication)**:

Why Is Replication Useful?

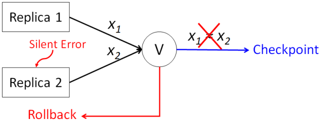
- **Error detection (duplication)**:



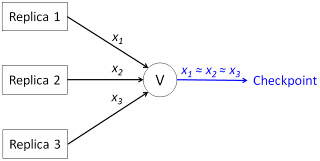
- **Error correction (triplication)**:

Why Is Replication Useful?

- **Error detection (duplication)**:

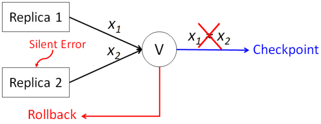


- **Error correction (triplication)**:

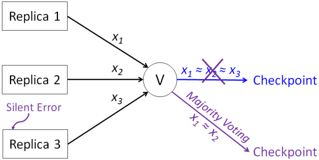


Why Is Replication Useful?

- **Error detection (duplication)**:

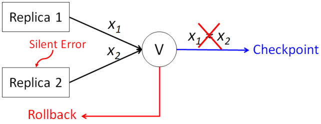


- **Error correction (triplication)**:

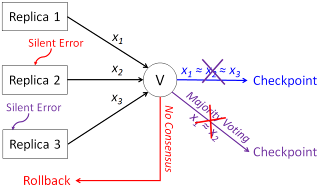


Why Is Replication Useful?

- **Error detection (duplication)**:

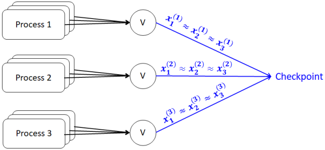


- **Error correction (triplication)**:

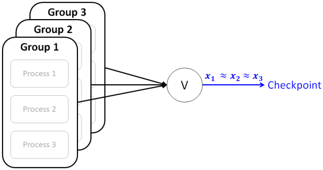


Two Replication Modes

- **Process Replication**:

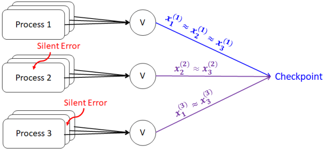

Process 1
Process 2
Process 3
V
V
V
$x_1^{(1)} \approx x_2^{(1)} \approx x_3^{(1)}$
$x_1^{(2)} \approx x_2^{(2)} \approx x_3^{(2)}$
$x_1^{(3)} \approx x_2^{(3)} \approx x_3^{(3)}$
Checkpoint

- **Group Replication**:

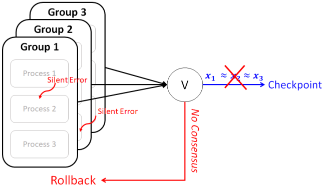


Two Replication Modes

- **Process Replication**:


Process 1
Silent Error
Process 2
Silent Error
Process 3
V
V
V
$x_1^{(1)} \approx x_2^{(1)} \approx x_3^{(1)}$
$x_2^{(2)} \approx x_3^{(2)}$
$x_1^{(3)} \approx x_3^{(3)}$
Checkpoint

- **Group Replication**:



A few questions

Silent errors

- Error rate? MTBE?
- Selective reliability?
- New algorithms beyond iterative? matrix-product, FFT, ...
- Multi-level patterns for both fail-stop and silent errors

Resilient research on resilience

Models needed to assess techniques at scale
without bias ☺

A few questions

Silent errors

- Error rate? MTBE?
- Selective reliability?
- New algorithms beyond iterative? matrix-product, FFT, ...
- Multi-level patterns for both fail-stop and silent errors

Resilient research on resilience

Models needed to assess techniques at scale
without bias ☺

Outline

1 Scheduling checkpoints

2 IO Contention
- Scheduling strategies
- Simulations

3 Replication for fail-stop failures

4 Scheduling Stochastic Tasks

5 Conclusion

IO contention

- Space-sharing prevalent in HPC platforms
- Application instances:
  - have dedicated computational nodes
  - share interconnect links and storage partition (PFS)
  - checkpoint (to stable storage) independently

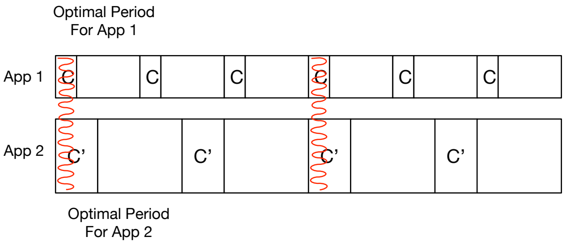
⇒ network and storage contention

Checkpointing 101 (revisited)

When do applications checkpoint on HPC systems?

- State-of-the-art: Young/Daly period
- Standard practice: every hour ☹

How long does it take to checkpoint?



- Optimal period computed assuming fixed checkpoint cost
- Interferences between checkpointing I/O of App 1 and App 2 change their checkpoint time
  $\Rightarrow$ Applications checkpoint too often

When to checkpoint in a shared environment,
since checkpoint cost is not predictable?

Model

Platform

- I/O subsystem time-shared (contended)
- Linear interference model

Workload

- Many applications but only a few classes (sets of applications with similar sizes, durations, footprints and I/O needs)
- Initialization and finalization I/O at max bandwidth; regular (non-CR) I/O evenly distributed over execution
- Job makespans known a priori
- Simulations based on APEX workflow / Cielo platform

Checkpoint

- Fixed: 1 hour (unless otherwise specified)
- Daly: uses Young/Daly application period $\sqrt{2C_{app}\mu_{app}}$

Outline

1. Scheduling checkpoints
2. IO Contention
   - Scheduling strategies
   - Simulations
3. Replication for fail-stop failures
4. Scheduling Stochastic Tasks
5. Conclusion

I/O Scheduling Algorithms: standard approaches

Oblivious (Fixed / Daly)

No scheduling of any I/O: when overlapping, interfere linearly

⇒ Risk of I/O Inefficiency

Ordered (Fixed / Daly)

I/O (checkpoint or init/final) served First Come - First Served

If another application is being served, wait in turn

⇒ Risk of delayed I/O and checkpoints, increasing waste

I/O Scheduling Algorithms: new approaches

Ordered-NB (Fixed / Daly)

I/O (checkpoint or init/final) served First Come - First Served

In case of checkpoints, continue working until served

⇒ Risk of extra re-execution due to delayed checkpoints

Least-Waste

Serve I/O request that minimizes potential waste

⇒ Checkpoints are non-blocking: continue working until they are served

⇒ Daly period embedded in scheduling (prevent from checkpointing too often)

Oblivious

- Jobs fill up the system based on processor availability
- I/O workloads (including CR activities) not coordinated
- Each I/O stream given decrease in bandwidth linearly proportional to the number of competing operations
- Subsequent checkpoint scheduled to start after $P_i - C_i$
  $\Rightarrow$ Resultant checkpoint period may be longer than P_i

Ordered

- Blocking FCFS I/O Scheduling
- I/O requests performed sequentially, in request arrival order
- Jobs with outstanding I/O requests blocked until their requests are completed
- With two jobs simultaneously requesting I/O of volume V_1, V_2:
    - *Oblivious*: Linear interference (both jobs I/O are slowed down) until the smallest of (V_1, V_2) is transferred
    - *Ordered*:
      - first scheduled job takes $\frac{V_1}{\beta_{\text{avail}}}$
      - second job waits $\frac{V_1}{\beta_{\text{avail}}}$ then takes $\frac{V_2}{\beta_{\text{avail}}}$
- Resultant checkpoint period may be longer than P_i

Ordered-NB

- Non-Blocking FCFS I/O Scheduling
- Refactor code to continue computing while awaiting **checkpoint** I/O
- Previous checkpoint ends at time t_{now}
  $\Rightarrow$ tentative time for next checkpoint $t_{req} = t_{now} + P_i - C_i$
- At t_{req}, make non-blocking I/O request (I/O token still FCFS)
- Job continues until I/O token is available
- At this point, job generates its checkpoint data
- Use existing APIs in SCR or FTI to regularly poll if a checkpoint should be taken at this time
- Postponed checkpoint ⇒ increased risk exposure

Least-Waste

- Non-Blocking least waste I/O Scheduling
- When an I/O request completes at time t, select best candidate from pool:
    - IO-CANDIDATE $\mathcal{C}_{IO}$
      Job J_i, $1 \leq i \leq r$ with an (input, output or recovery) I/O request of length v_i seconds, has q_i processors, initiated its I/O request d_i seconds ago (idle since)
    - CKPT-CANDIDATE $\mathcal{C}_{Ckpt}$
      Job J_i, $r+1 \leq i \leq r+s$, with a checkpoint duration of C_i seconds and q_i processors, took its last checkpoint d_i seconds ago and keeps executing, with $d_i \geq P_{Daly}(J_i)$

Job selection

- $J_i \in \mathcal{C}_{IO}$ uses the I/O resource for v_i seconds
  - For $J_j \in \mathcal{C}_{IO}$, $W_i(j) = q_j(d_j + v_i)$
  - For $J_j \in \mathcal{C}_{Ckpt}$, $W_i(j) = \frac{v_i}{\mu_{\text{ind}}} q_j^2 (R_j + d_j + \frac{v_i}{2})$
  - Expected waste $W_i = \sum_{J_j \in \mathcal{C}_{IO}, j \neq i} W_i(j) + \sum_{J_j \in \mathcal{C}_{Ckpt}} W_i(j)$
- $J_i \in \mathcal{C}_{Ckpt}$ uses the I/O resource for C_i seconds
  - Similar equations ...
- Select job $J_i \in \mathcal{C}_{IO} \cup \mathcal{C}_{Ckpt}$ whose waste W_i is minimal

Feasibility of Cooperative Strategies

- *Ordered*, *Ordered-NB*, *Least-Waste* require synchronization
- *Ordered*

  at filesystem level
- *Ordered-NB* and *Least-Waste*:

  modify apps to continue working until access is granted

  ⇒ implementation in checkpointing library SCR or FTI
- Memory hierarchy:

  - checkpoint process memory on unreliable (but fast) media

  - upload checkpoints in the background,

  while the application proceeds to compute

Steady-state

- n_i jobs of class A_i, q_i nodes, $C_i = \frac{size_i}{\beta_{\text{avail}}}$
- Waste of J_i with checkpoint period P_i:

$$W_i = W_i(P_i) = \frac{C_i}{P_i} + \frac{q_i}{\mu}(\frac{P_i}{2} + R_i)$$

MINIMIZE

$$W = \sum_i \frac{n_i q_i}{\mathcal{N}} \left(\frac{C_i}{P_i} + \frac{q_i}{\mu}(\frac{P_i}{2} + R_i) \right)$$

SUBJECT TO

$$F = \sum_i \frac{n_i C_i}{P_i} \leq 1$$

Lower bound

KKT

$$P_i = \sqrt{\frac{2\mu\mathcal{N}}{q_i^2}\left(\frac{q_i}{\mathcal{N}} + \lambda\right) C_i}$$

- Choose λ minimal s.t. $F \leq 1$ (solve numerically)
- $\lambda = 0 \Rightarrow$ Young/Daly
- I/O constraint not sufficient
  $\Rightarrow$ orchestrate checkpoints into periodic repeating pattern
  $\Rightarrow$ lower bound of $W = \sum_i \frac{n_i q_i}{\mathcal{N}} W_i(P_i)$

Outline

1. Scheduling checkpoints
2. IO Contention
   - Scheduling strategies
   - Simulations
3. Replication for fail-stop failures
4. Scheduling Stochastic Tasks
5. Conclusion

LANL Workloads from the APEX Workflows report

Workflow	EAP	LAP	Silverton	VPIC
Workload percentage	66	5.5	16.5	12
Work time (h)	262.4	64	128	157.2
Number of cores	16384	4096	32768	30000
Initial Input (% of memory)	3	5	70	10
Final Output (% of memory)	105	220	43	270
Checkpoint Size (% of memory)	160	185	350	85

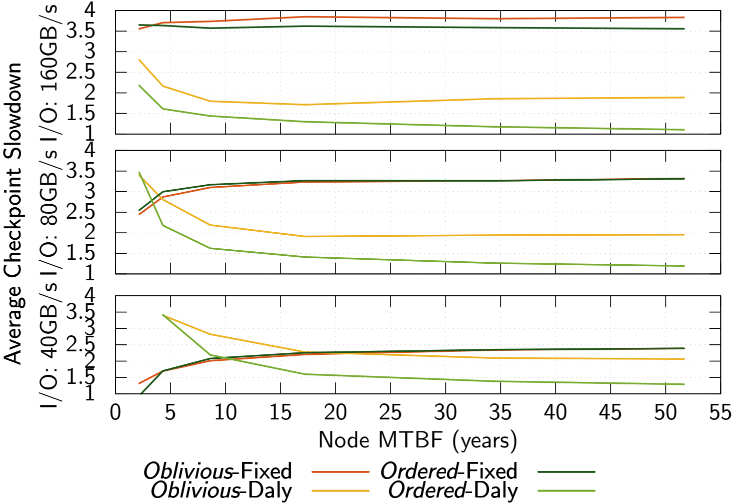
Cielo

- 1.37 Petaflops capability system at LANL (2010-2016)
- 143,104 cores, 286 TB main memory
- PFS with theoretical maximum capacity 160GB/s

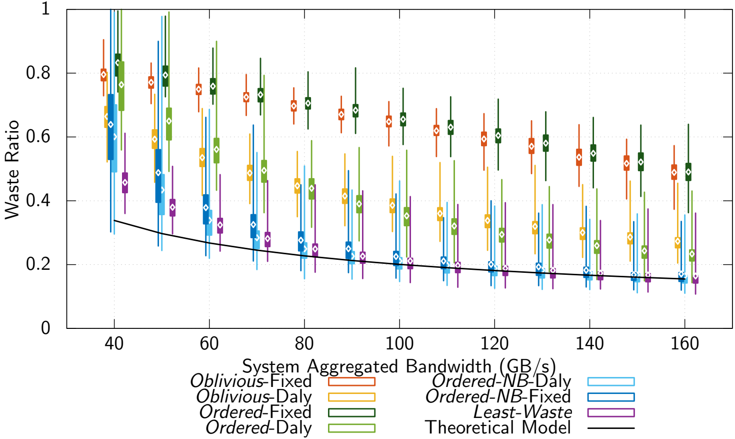
Simulation Framework

- Random selection of jobs according to class ratios
- Duration uniformly distributed between $0.8w$ and $1.2w$
- Generation of node failures with Exponential distributions
- First-fit strategy (job characteristics, job priority, resource availability)
- Simulate online scheduling
- Restarted jobs set to highest priority

Slowdown of checkpoints

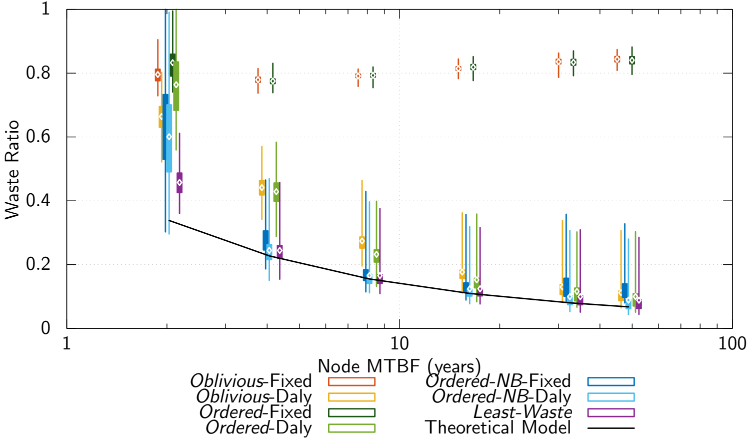


Waste as a function of system bandwidth



$$\mu_{\text{ind}} = 2\text{years},\ \mu = 1\text{hour}$$

Waste as a function of system MTBF

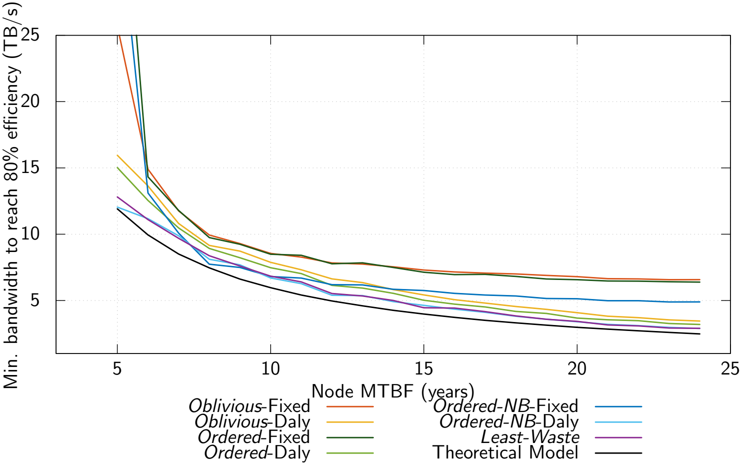


$\beta_{\text{avail}} = 40\text{GBs}$

Prospective system (1/2)

- Aurora-like
  7PB of main memory and 50,000 compute nodes
- Scale APEX workflow
  accordingly to Aurora/Celio memory size increase

Prospective system (2/2)



Minimum aggregated filesystem bandwidth to reach 80% efficiency

Burst buffers

- Dedicated
  - Same throughput constraint
  - Schedule according to priority
  - Allows for some slack (shift checkpoints)
- Shared
  - Hierarchical system
  - Same contention problem at subsystem level
- See IJNC paper, 2019. Also RR Inria 9109

Takeaway

- Checkpoint/restart:
  - standard for fault-protection on production platforms
  - increases the burden of the already overtaxed I/O subsystem
- Cooperative strategies outperform selfish approaches w.r.t. platform utilization
- Trade-off between platform utilization and worst time to completion of individual applications

Outline

1. Scheduling checkpoints
2. IO Contention
3. Replication for fail-stop failures
4. Scheduling Stochastic Tasks
5. Conclusion

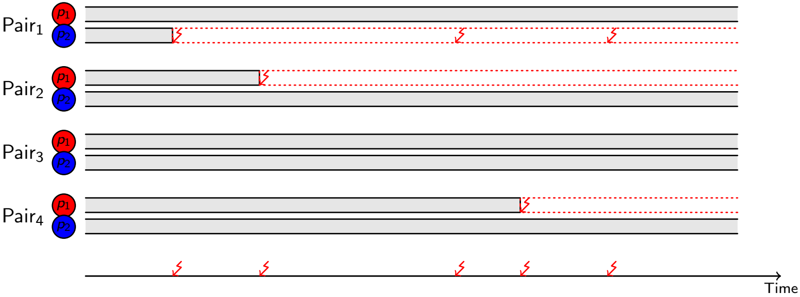
Replication

- Full replication: efficiency $< 50\%$
- Can replication+checkpointing be more efficient than checkpointing alone?
- Study by Ferreira et al. [SC'2011]: yes
- Revisited by Hussain, Znati and Melhme [SC'2018]: yes

Model by Ferreira et al. [SC' 2011]

- Platform with $N = 2b$ processors arranged into b pairs
- Parallel application with b processes, each replicated
- When a replica is hit by a failure, it is not restarted
- Application fails when both replicas in one pair have been hit

Example



Mean Time To Interruption

- $n_{\text{fail}}(2b)$ expected number of failures to interrupt the applications
- MTTI M_{2b} = Mean Time to Interruption
  $\Rightarrow$ replaces MTBF from the application perspective

$$M_{2b} = n_{\text{fail}}(2b) \times \mu_{2b} = n_{\text{fail}}(2b) \times \frac{\mu}{2b} \tag{1}$$

Proposition

$$n_{fail}(2b) = 1 + 4^b \;/\; \binom{2b}{b} \approx \sqrt{\pi b}$$

Checkpointing

$$\text{No Replication} \quad T_{YD} = \sqrt{2\mu_N C} \tag{2}$$

$$\text{Full Replication} \quad T_{MTTI} = \sqrt{2M_N C} \tag{3}$$

What's wrong?

$$T_{MTTI} = \sqrt{2 M_N C}$$

- Just an approximation. How accurate?
- Risk is increasing as more and more processors die until application crash
  ⇒ Periodic checkpointing (most likely) not optimal ☹

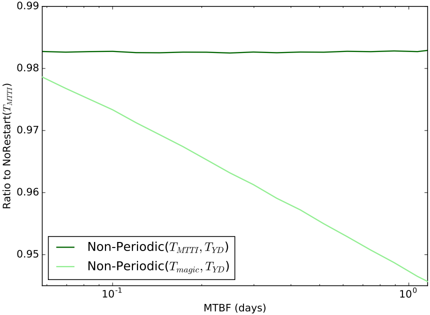
With a single processor pair

- With one processor: $T_{YD} = \sqrt{2\mu C}$
- With replication: $n_{\text{fail}}(2) = 3$, $M_2 = 3\frac{\mu}{2}$, $T_{MTTI} = \sqrt{3\mu C}$
- Magic period: $T_{magic} = \left(\frac{3}{4}C\mu^2\right)^{\frac{1}{3}}$

Three variants:

- Periodic with period T_{MTTI}: baseline
- NonPeriodic(T_1, T_2):
  - use T_1 while both processors are alive
  - switch to T_2 at checkpoint after first failure
    - Variant 1: $T_1 = T_{MTTI}$, $T_2 = T_{YD}$
    - Variant 2: $T_2 = T_{magic}$, $T_2 = T_{YD}$
- 100,000 simulations, each with 10,000 periods

With a single processor pair



Ratio of time to solution of two non-periodic strategies over time-to-solution of periodic approach, with $C = 60$

$$\mu = 10 \text{ hours}$$
$$\Rightarrow T_{YD} = 34.6mn, T_{MTTI} = 42.4mn, T_{magic} = 64.6mn$$

Takeaway

- Opinion is divided about replication
- If needed, use it as efficiently as possible
- Best checkpoint strategy with many processor pairs?

Outline

1 Scheduling checkpoints

2 IO Contention

3 Replication for fail-stop failures

4 Scheduling Stochastic Tasks
- Simple instance
- Experiments

5 Conclusion

A little scheduling problem

- Independent tasks, IID execution times with distribution $\mathcal{D}$
- Platform: identical processors, unit speed, unit cost
- User: limited budget b and execution deadline d
- Objective: maximize expected number of tasks completed

Motivation

Imprecise computations: tasks have a mandatory part and optional part, maximize optional parts with leftover time and budget

A little scheduling problem (2/2)

Scheduling policy

- Decide how many processors to launch & stop at each second
- Processors interrupted when deadline or budget is exceeded
- Each task can be deleted at any instant before completion
- Non-preemptive execution:
  - interrupted tasks cannot be relaunched
  - time/budget spent until interruption: completely lost

Outline

1 Scheduling checkpoints

2 IO Contention

3 Replication for fail-stop failures

4 Scheduling Stochastic Tasks
- Simple instance
- Experiments

5 Conclusion

Simple instance

- One processor
- Unlimited budget, no deadline
- Discrete distribution:

Probability	Execution time
p_1	w_1
p_2	w_2
p_3	w_3

- Objective: maximize success rate per time/budget unit $\mathcal{R}$

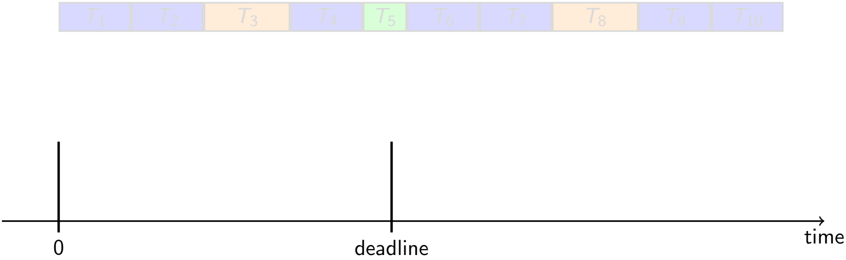
Simple instance

- One processor
- Unlimited budget, no deadline
- Discrete distribution:

Probability	Execution time
$p_1 = 0.1$	$w_1 = 3$
$p_2 = 0.7$	$w_2 = 5$
$p_3 = 0.2$	$w_3 = 6$

- Objective: maximize success rate per time/budget unit $\mathcal{R}$

Illustrating example

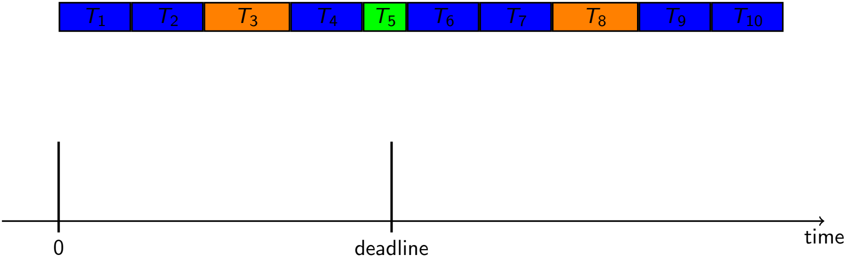


Never interrupt tasks: 4 tasks completed.

Interrupt tasks after w_1: 1 task completed.

Interrupt tasks after w_2: 4 tasks completed.

Illustrating example

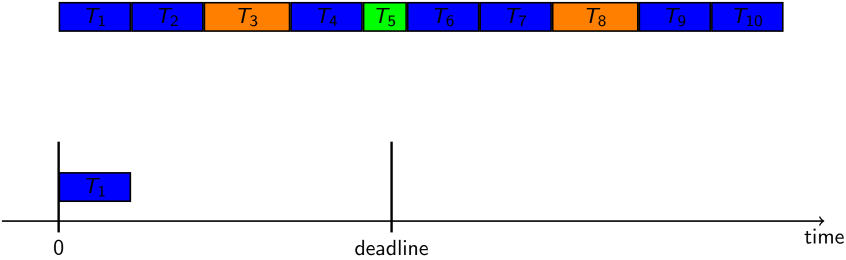


Never interrupt tasks: 4 tasks completed.

Interrupt tasks after w_1: 1 task completed.

Interrupt tasks after w_2: 4 tasks completed.

Illustrating example

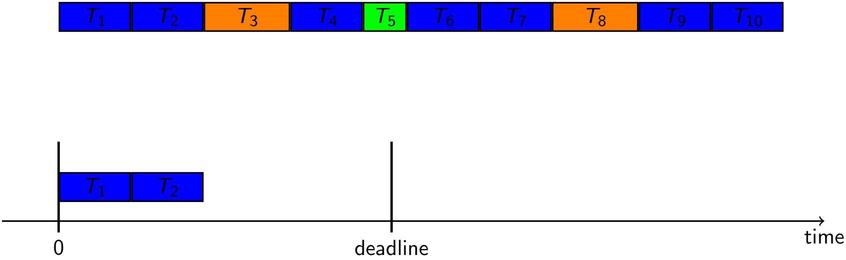


Never interrupt tasks: 4 tasks completed.

Interrupt tasks after w_1: 1 task completed.

Interrupt tasks after w_2: 4 tasks completed.

Illustrating example

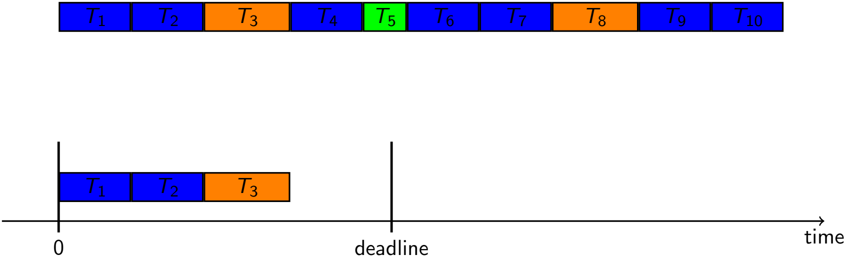


Never interrupt tasks: 4 tasks completed.

Interrupt tasks after w_1: 1 task completed.

Interrupt tasks after w_2: 4 tasks completed.

Illustrating example

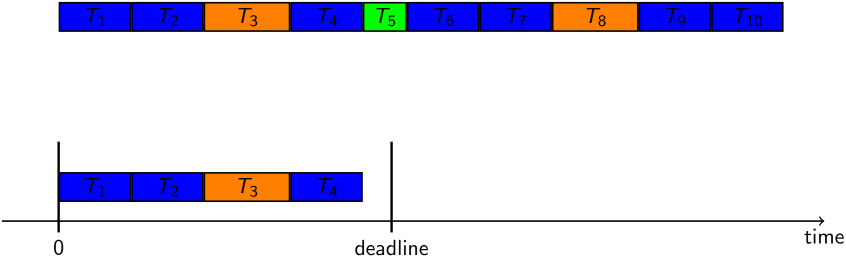

T_1 T_2 T_3 T_4 T_5 T_6 T_7 T_8 T_9 T_{10}
T_1 T_2 T_3
0
deadline
time

Never interrupt tasks: 4 tasks completed.

Interrupt tasks after w_1: 1 task completed.

Interrupt tasks after w_2: 4 tasks completed.

Illustrating example

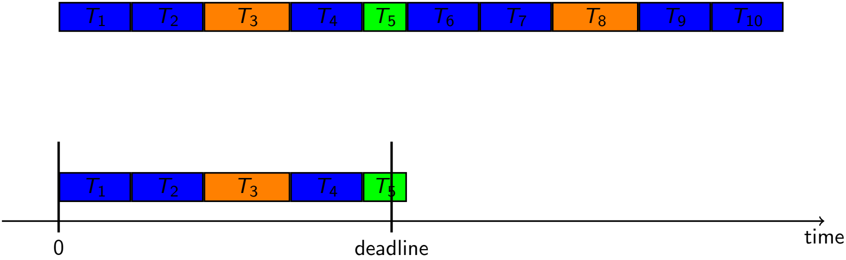


Never interrupt tasks: 4 tasks completed.

Interrupt tasks after w_1: 1 task completed.

Interrupt tasks after w_2: 4 tasks completed.

Illustrating example

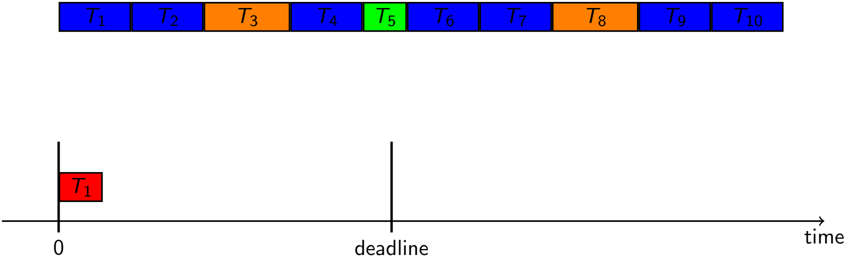


Never interrupt tasks: 4 tasks completed.

Interrupt tasks after w_1: 1 task completed.

Interrupt tasks after w_2: 4 tasks completed.

Illustrating example

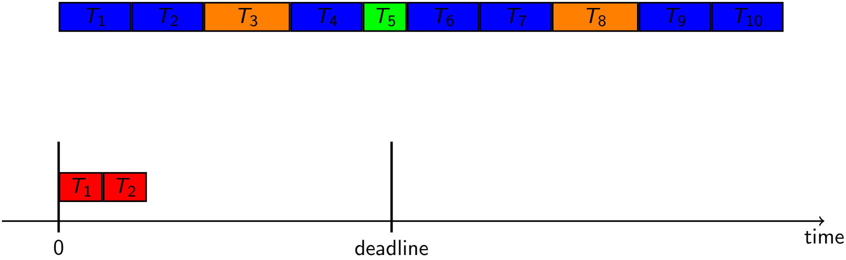


Never interrupt tasks: 4 tasks completed.

Interrupt tasks after w_1: 1 task completed.

Interrupt tasks after w_2: 4 tasks completed.

Illustrating example

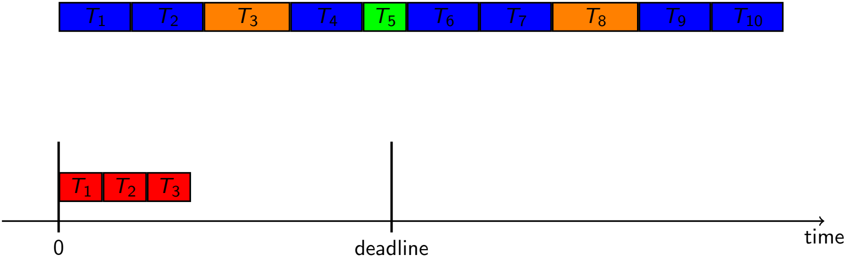


Never interrupt tasks: 4 tasks completed.

Interrupt tasks after w_1: 1 task completed.

Interrupt tasks after w_2: 4 tasks completed.

Illustrating example

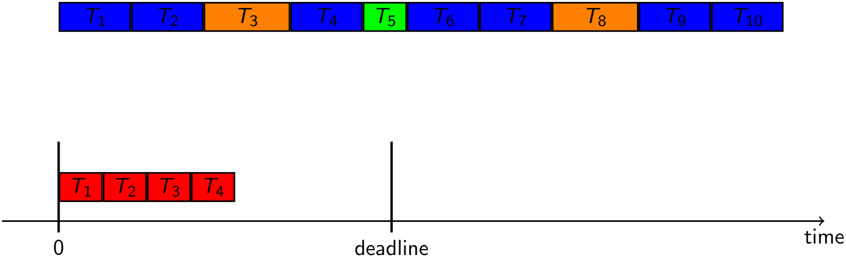


Never interrupt tasks: 4 tasks completed.

Interrupt tasks after w_1: 1 task completed.

Interrupt tasks after w_2: 4 tasks completed.

Illustrating example

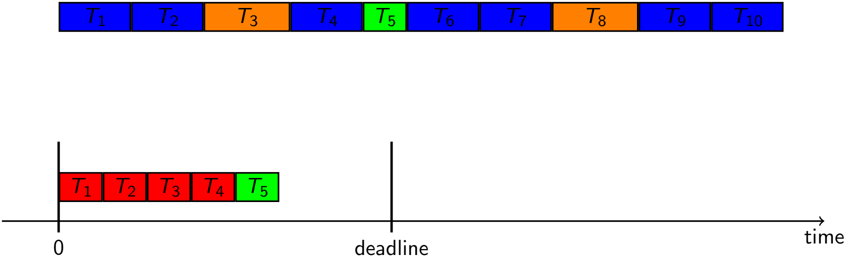


Never interrupt tasks: 4 tasks completed.

Interrupt tasks after w_1: 1 task completed.

Interrupt tasks after w_2: 4 tasks completed.

Illustrating example

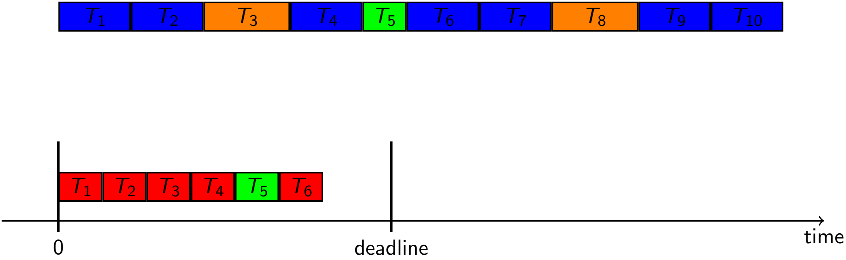


Never interrupt tasks: 4 tasks completed.

Interrupt tasks after w_1: 1 task completed.

Interrupt tasks after w_2: 4 tasks completed.

Illustrating example

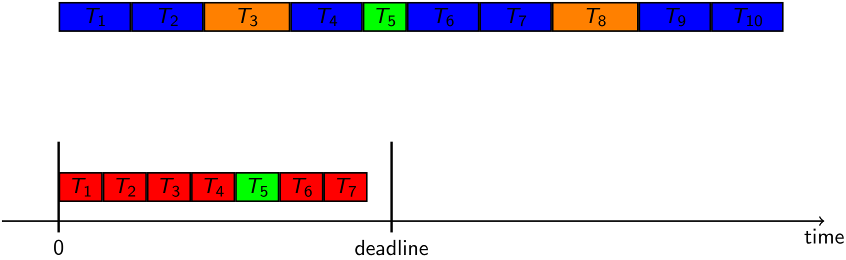


Never interrupt tasks: 4 tasks completed.

Interrupt tasks after w_1: 1 task completed.

Interrupt tasks after w_2: 4 tasks completed.

Illustrating example

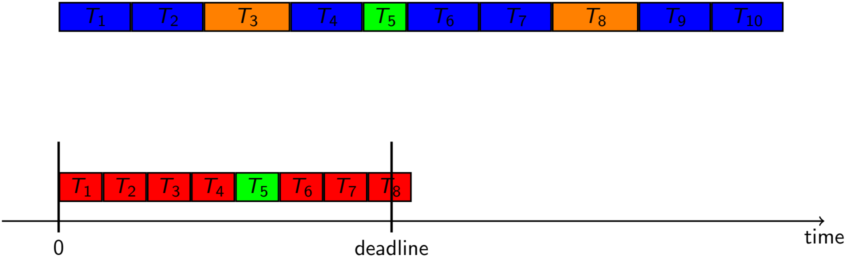


Never interrupt tasks: 4 tasks completed.

Interrupt tasks after w_1: 1 task completed.

Interrupt tasks after w_2: 4 tasks completed.

Illustrating example

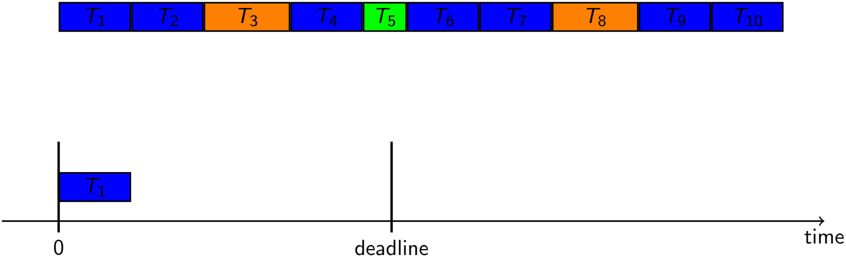

T_1 T_2 T_3 T_4 T_5 T_6 T_7 T_8 T_9 T_{10}
T_1 T_2 T_3 T_4 T_5 T_6 T_7 T_8
0
deadline
time

Never interrupt tasks: 4 tasks completed.

Interrupt tasks after w_1: 1 task completed.

Interrupt tasks after w_2: 4 tasks completed.

Illustrating example

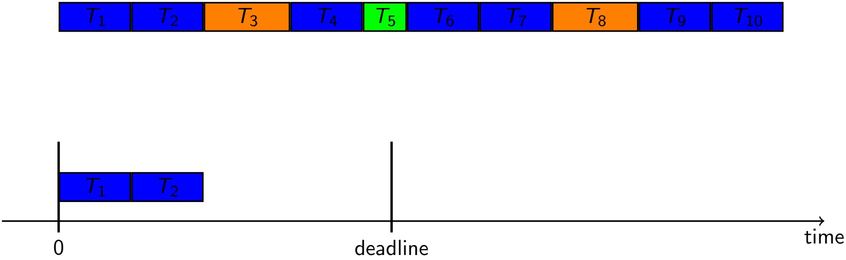


Never interrupt tasks: 4 tasks completed.

Interrupt tasks after w_1: 1 task completed.

Interrupt tasks after w_2: 4 tasks completed.

Illustrating example

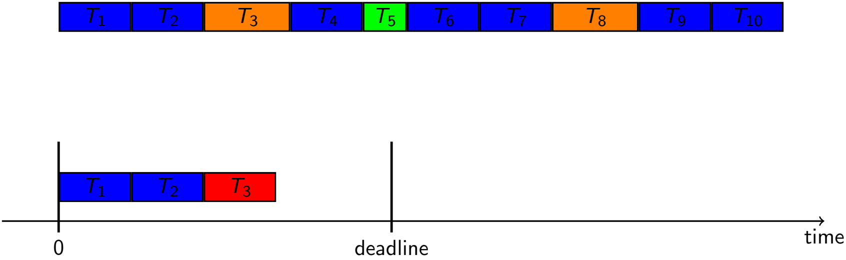


Never interrupt tasks: 4 tasks completed.

Interrupt tasks after w_1: 1 task completed.

Interrupt tasks after w_2: 4 tasks completed.

Illustrating example

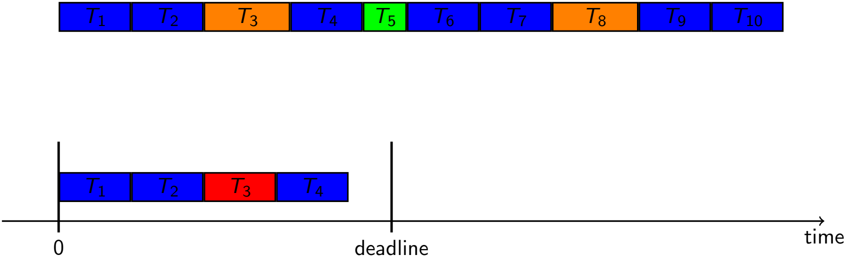


Never interrupt tasks: 4 tasks completed.

Interrupt tasks after w_1: 1 task completed.

Interrupt tasks after w_2: 4 tasks completed.

Illustrating example

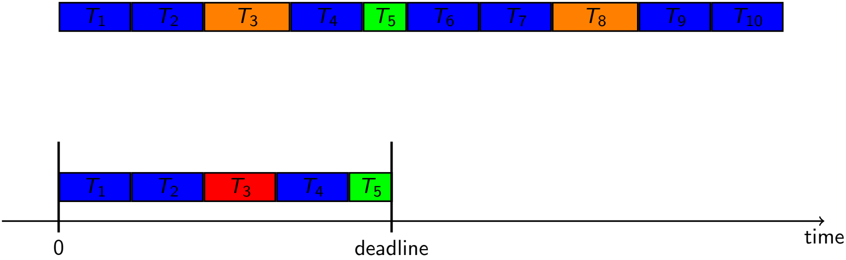


Never interrupt tasks: 4 tasks completed.

Interrupt tasks after w_1: 1 task completed.

Interrupt tasks after w_2: 4 tasks completed.

Illustrating example



Never interrupt tasks: 4 tasks completed.

Interrupt tasks after w_1: 1 task completed.

Interrupt tasks after w_2: 4 tasks completed.

Scheduling strategies

Probability	Execution time
$p_1 = 0.1$	$w_1 = 3$
$p_2 = 0.7$	$w_2 = 5$
$p_3 = 0.2$	$w_3 = 6$

- Stop all tasks after w_1: $\mathcal{R}_1 = \frac{p_1}{w_1} = \frac{1}{30}$
-
-

Scheduling strategies

Probability	Execution time
$p_1 = 0.1$	$w_1 = 3$
$p_2 = 0.7$	$w_2 = 5$
$p_3 = 0.2$	$w_3 = 6$

- Stop all tasks after w_1: $\mathcal{R}_1 = \frac{p_1}{w_1} = \frac{1}{30}$
- Stop all tasks after w_2: $\mathcal{R}_2 = \frac{p_1+p_2}{p_1 w_1+(1-p_1)w_2} = \frac{1}{6}$
-

Scheduling strategies

Probability	Execution time
$p_1 = 0.1$	$w_1 = 3$
$p_2 = 0.7$	$w_2 = 5$
$p_3 = 0.2$	$w_3 = 6$

- Stop all tasks after w_1: $\mathcal{R}_1 = \frac{p_1}{w_1} = \frac{1}{30}$
- Stop all tasks after w_2: $\mathcal{R}_2 = \frac{p_1+p_2}{p_1 w_1+(1-p_1)w_2} = \frac{1}{6}$
- Stop half unsuccessful tasks after w_1 and one-third after w_2: $\mathcal{R} = ?$

Optimal strategy

Theorem

Best strategy is to stop all tasks at some threshold

Strategy

Find *i* maximizing

$$\mathcal{R}_i \stackrel{\text{def}}{=} \frac{\sum_{j=1}^{i} p_j}{\sum_{j=1}^{i} p_j w_j + (1 - \sum_{j=1}^{i} p_j) w_i}$$

If ties, pick smallest index

Scheduling strategies

Probability	Execution time
$p_1 = 0.1$	$w_1 = 3$
$p_2 = 0.7$	$w_2 = 5$
$p_3 = 0.2$	$w_3 = 6$

- Stop all tasks after w_1: $\mathcal{R}_1 = \frac{p_1}{w_1} = \frac{1}{30}$
- Stop all tasks after w_2: $\mathcal{R}_2 = \frac{p_1+p_2}{p_1 w_1+(1-p_1)w_2} = \frac{1}{6}$
- Stop all tasks after w_3: $\mathcal{R}_3 = \frac{1}{p_1 w_1+p_2 w_2+p_3 w_3} = \frac{1}{5}$

Scheduling strategies

Probability	Execution time
$p_1 = 0.1$	$w_1 = 3$
$p_2 = 0.7$	$w_2 = 5$
$p_3 = 0.2$	$w_3 = 6$

Question?

So in the end you should not interrupt anything, right?

Pfhhh these scheduling guys ☹

- Stop all tasks after w_2: $\mathcal{R}_2 = \frac{p_1+p_2}{p_1 w_1+(1-p_1)w_2} = \frac{1}{6}$
- Stop all tasks after w_3: $\mathcal{R}_3 = \frac{1}{p_1 w_1+p_2 w_2+p_3 w_3} = \frac{1}{5}$

Scheduling strategies

Probability	Execution time
$p_1 = 0.1$	$w_1 = 3$
$p_2 = 0.7$	$w_2 = 5$
$p_3 = 0.2$	$w_3 = 101$

- Stop all tasks after w_1: $\mathcal{R}_1 = \frac{p_1}{w_1} = \frac{1}{30}$
- Stop all tasks after w_2: $\mathcal{R}_2 = \frac{p_1+p_2}{p_1 w_1+(1-p_1)w_2} = \frac{1}{6}$
- Stop all tasks after w_3: $\mathcal{R}_3 = \frac{1}{p_1 w_1+p_2 w_2+p_3 w_3} = \frac{1}{24}$

From discrete to continuous distributions

- $f(x)$ probability density, $F(x)$ cumulative distribution
- Expected value μ_D, variance, σ_D^2

$$\arg\max_i \mathcal{R}_i \overset{\text{def}}{=} \frac{\sum_{j=1}^{i} p_j}{\sum_{j=1}^{i} p_j w_j + (1 - \sum_{j=1}^{i} p_j) w_i}$$

$$\arg\max_l \mathcal{R}(l) \overset{\text{def}}{=} \frac{F(l)}{\int_0^l x f(x) dx + (1 - F(l)) l}$$

No more a theorem, but hopefully a good heuristic ...

Best cutting threshold

- $\mathcal{D} = \textsc{Exp}(\lambda)$
-

Best cutting threshold

- $\mathcal{D} = \textsc{Exp}(\lambda)$
  Interrupt at any instant ($\mathcal{R}_I$ constant)

- $\mathcal{D} = \textsc{Uniform}[a, b]$

Best cutting threshold

- $\mathcal{D} = \textsc{Exp}(\lambda)$
  Interrupt at any instant ($\mathcal{R}_l$ constant)

- $\mathcal{D} = \textsc{Uniform}[a, b]$
  Never interrupt ($\mathcal{R}_l$ maximal for $l = b$)

Best cutting threshold

Question?

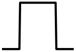
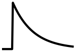
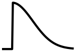
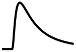
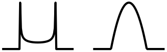
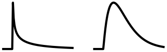
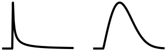
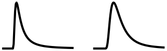
Well, do you know any important distribution for which it is really worth interrupting tasks?

Pfhhh these scheduling guys ☹

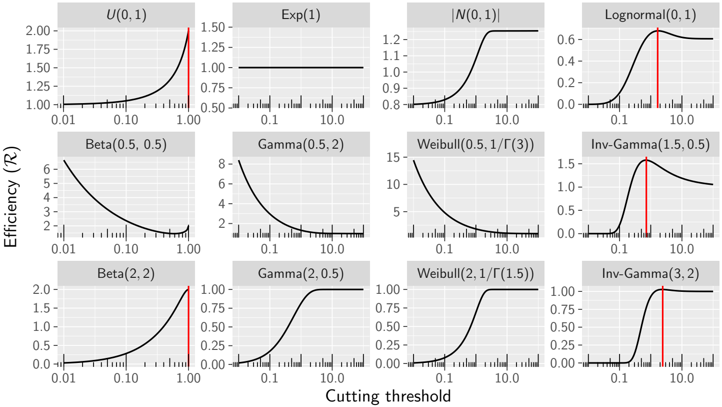
- $\mathcal{D} = \text{UNIFORM}[a, b]$

  Never interrupt ($\mathcal{R}_l$ maximal for $l = b$)

A few distributions . . .

Name	PDF	Density
Uniform	$\frac{1}{b-a}$	
Exponential	$\lambda e^{-\lambda x}$	
Half-normal	$\frac{\sqrt{2}}{\theta\sqrt{\pi}} e^{-\frac{x^2}{2\theta^2}}$	
Lognormal	$\frac{1}{x\beta\sqrt{2\pi}} e^{-\frac{(\log(x)-\alpha)^2}{2\beta^2}}$	
Beta	$\frac{x^{\alpha-1}(1-x)^{\beta-1}}{B(\alpha,\beta)}$	
Gamma	$\frac{1}{\Gamma(k)\theta^k} x^{k-1} e^{-\frac{x}{\theta}}$	
Weibull	$\frac{k}{\theta^k} x^{k-1} e^{-(\frac{x}{\theta})^k}$	
Inverse-gamma	$\frac{\theta^k}{\Gamma(k)} x^{-k-1} e^{-\frac{\theta}{x}}$	

. . . and their optimal cutting threshold



Outline

1 Scheduling checkpoints

2 IO Contention

3 Replication for fail-stop failures

4 Scheduling Stochastic Tasks
- Simple instance
- Experiments

5 Conclusion

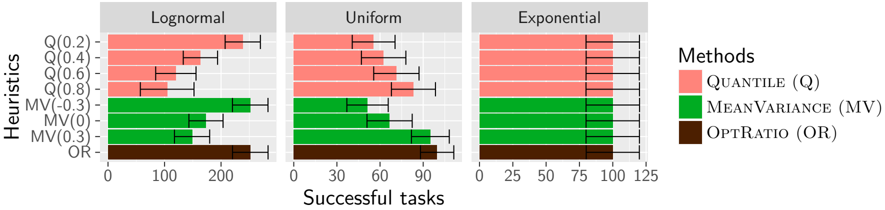
Heuristics with 1 processor

- MEANVARIANCE(x): kill a task as time $\mu_D + x\sigma_D$, with x some constant
- QUANTILE(x): kill a task when execution time reaches the x-quantile of $\mathcal{D}$, with $0 \leq x \leq 1$
- OPTRATIO: optimal cutting threshold

Heuristics with many processors

- With budget b and deadline d, enroll $\lceil \frac{b}{d} \rceil$ processors
- Run previous heuristics in parallel

Experiments



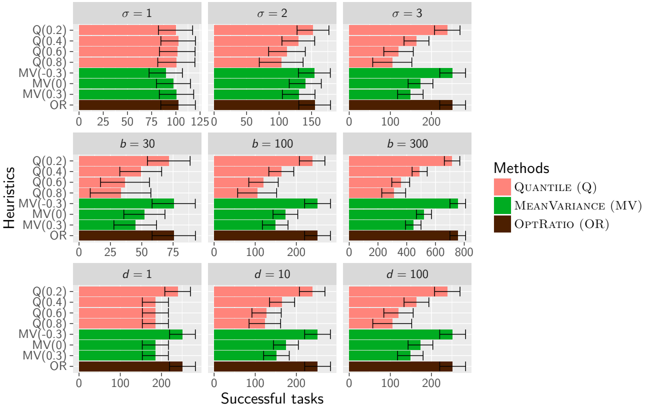
Normalized for $\mu = 1$, budget and deadline $b = d = 100$

Exponential: $\lambda = 1$, $l_{\text{opt}} = 2$ (arbitrarily)

Uniform: $a = 0$, $b = 2$, $l_{\text{opt}} = 2$

Lognormal: $\alpha \approx -1.15$, $\beta \approx 1.52$, $\mu = 1$, $\sigma = 3$, $l_{\text{opt}} \approx 0.1$

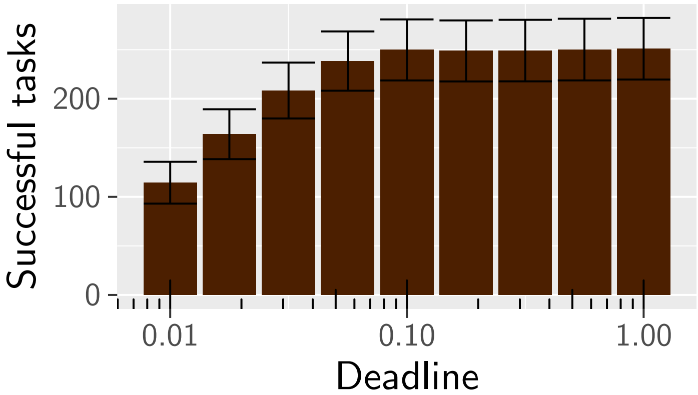
Focus on LogNormal



Lognormal: $\alpha \approx -1.15$, $\beta \approx 1.52$, $\mu = 1$, $\sigma = 3$, $l_{\text{opt}} \approx 0.1$

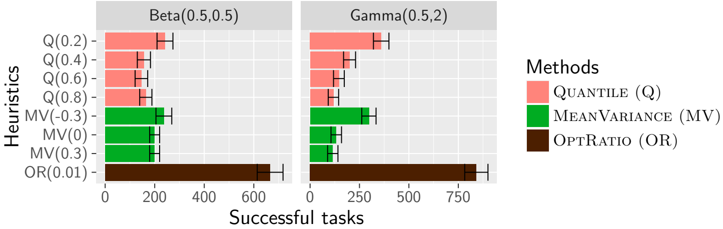
First row $b = d = 100$, second row $b = d$, third row $b = 100$ (hence $\lceil \frac{b}{d} \rceil$ processors)

With small deadlines



Budget $b = 100$, varying deadline (hence number of processors)
Lognormal: $\alpha \approx -1.15$, $\beta \approx 1.52$, $\mu = 1$, $\sigma = 3$, $l_{\text{opt}} \approx 0.1$

Cut them short

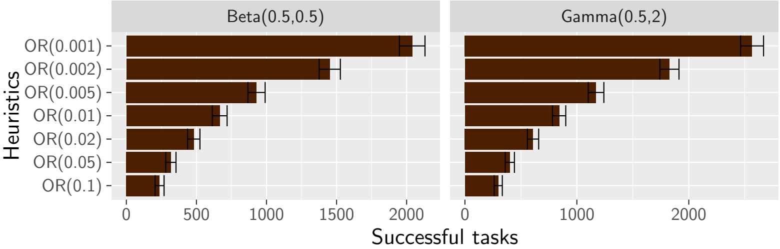


$\mu = 0.5$ for Beta, $\mu = 1$ for Gamma

cutting threshold is 0.01 for OR in both plots

$b = d = 100$

Zoom on threshold impact



$\mu = 0.5$ for $\textsc{Beta}$, $\mu = 1$ for $\textsc{Gamma}$

$b = d = 100$

Nice little (albeit technical) example

Probability	Execution time
$p_1 = 0.4$	$w_1 = 2$
$p_2 = 0.15$	$w_2 = 3$
$p_3 = 0.45$	$w_3 = 7$

Budget $b = 6$, no deadline (say $d = 6$)

Optimal schedule with 1 processor

Probability	Execution time
$p_1 = 0.40$	$w_1 = 2$
$p_2 = 0.15$	$w_2 = 3$

- $E(1) = 0,\ E(2) = p_1 = 0.4$
- $E(3) = (p_1 + p_2) = 0.55$ (pointless to kill an unsuccessful task at time 2)
- $E(4) = \max\{p_1 + E(2), p_1(1 + E(2)) + p_2(1 + E(1)) + p_3(0 + E(1))\} = 0.8$
  Either kill the first task (if not completed) at time 2
  or continue up to time 3 (if not completed) and then kill
- $E(6) = \max\{p_1 + E(4), p_1(1 + E(4)) + p_2(1 + E(3))\} = 1.2$

An efficient schedule with 2 processors

Probability	Execution time
$p_1 = 0.40$	$w_1 = 2$
$p_2 = 0.15$	$w_2 = 3$

- Two processors, each starting a task in parallel
- If none completes by time 2, let them run up to time 3
- Otherwise, kill at time 2 any not-yet completed task and start a new task

Processor 1

Processor 2

	w_1	w_2	w_3
w_1	$2 + p_1$	$1 + p_1$	$1 + p_1$
w_2	$1 + p_1$	2	1
w_3	$1 + p_1$	1	0

With probability $p_1 p_2$, 1st task completes, 2nd task is killed, 2 units remain for the newnone, expected number of completed tasks in this configuration is $1 + p_1$

$$E_{//} = p_1^2(2 + p_1) + 2p_1(p_2 + p_3)(1 + p_1) + 2p_2^2 + 2p_2 p_3 = 1.236$$

An efficient schedule with 2 processors

Probability	Execution time
$p_1 = 0.40$	$w_1 = 2$
$p_2 = 0.15$	$w_2 = 3$

- Two processors, each starting a task in parallel
- If none completes by time 2, let them run up to time 3
- Otherwise, kill at time 2 any not-yet completed task and start a new task

Question?

No kidding? You win 0.036 tasks in the end
and you are proud of you?!
Time to finish your talk!
Pfhhh these scheduling guys ☹

Outline

1 Scheduling checkpoints

2 IO Contention

3 Replication for fail-stop failures

4 Scheduling Stochastic Tasks

5 Conclusion

Conclusion

This talk

A few (simple) scheduling problems

Conclusion

This talk

A few (simple) scheduling problems

Future work

Multi-criteria scheduling problems

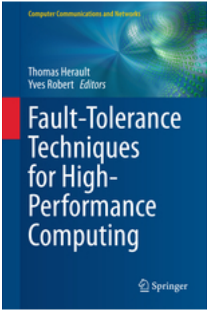
execution time/energy/reliability

add replication

best resource usage (performance trade-offs)

Several challenging algorithmic/scheduling problems ☺

Bibliography



Computer Communications and Networks

Thomas Herault
Yves Robert *Editors*

Fault-Tolerance Techniques for High-Performance Computing

Springer

First chapter = comprehensive survey, freely available
LAWN 289 (LApack Working Note)

Thanks

-

  Lyon: Anne Benoit, Louis-Claude Canon, Aurélien Cavelan, Aurélie Kong Win Chang, Valentin Le Fèvre, Frédéric Vivien

-

  Knoxville: George Bosilca, Aurélien Bouteiller, Jack Dongarra, Thomas Herault

-

  And: Dorian Arnold (Emory), Franck Cappello (Argonne), Kurt Ferreira (Sandia), Hongyang Sun (Vanderbilt)