Optimal resilience patterns to cope with fail-stop and silent errors

Anne Benoit[1] Aurélien Cavelan[1]
Yves Robert[1,2] Hongyang Sun[1]

[1]ENS Lyon & INRIA, France.

[2]University of Tennessee Knoxville, USA.

aurelien.cavelan@inria.fr

May 19, 2016, Nashville.

Why resilience?

Computing at exascale

- Larger node count: 10^5 or 10^6 nodes, each with 10^2 or 10^3 cores
- Shorter Mean Time Between Failures (MTBF) μ

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

MTBF (individual node)	1 year	10 years	100 years
MTBF (platform of 10^6 nodes)	30 secs	5 mins	50 mins

Multiple error sources

- Many papers address fail-stop errors
- Many others address silent errors (or silent data corruptions)

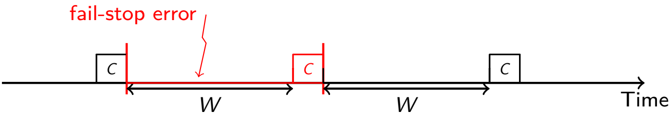
HPC applications must cope with **both** error sources! ☹

Objective: unified framework and optimal algorithmic solutions ☺

Coping with fail-stop errors

Instantaneous error detection, e.g., resource crash

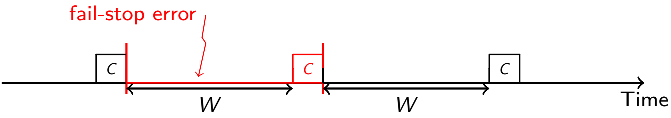
Standard approach: Periodic checkpoint, rollback, and recovery:



Coping with fail-stop errors

Instantaneous error detection, e.g., resource crash

Standard approach: Periodic checkpoint, rollback, and recovery:



General-purpose approach! ☺

Theorem.

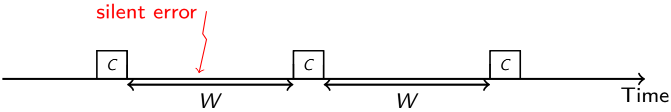
$$W^* = \sqrt{2\mu C} \quad \text{[Young 1974, Daly, 2006]}$$

μ: Platform MTBF
C: Checkpointing time

Coping with Silent Errors

Silent error detected only when corrupted data is activated
e.g., soft faults in L1 cache, ALU, double bit flip.

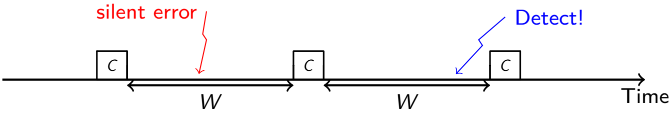
Main problem: detection latency
Same approach?



Coping with Silent Errors

Silent error detected only when corrupted data is activated
e.g., soft faults in L1 cache, ALU, double bit flip.

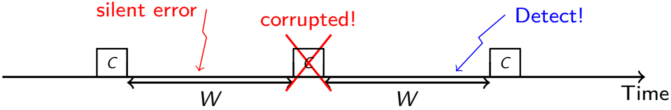
Main problem: detection latency
Same approach?



Coping with Silent Errors

Silent error detected only when corrupted data is activated
e.g., soft faults in L1 cache, ALU, double bit flip.

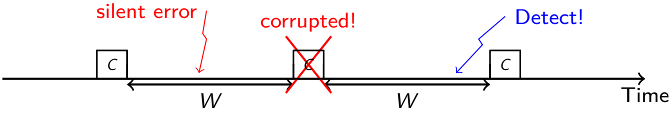
Main problem: detection latency
Same approach?



Coping with Silent Errors

Silent error detected only when corrupted data is activated
e.g., soft faults in L1 cache, ALU, double bit flip.

Main problem: detection latency
Same approach?

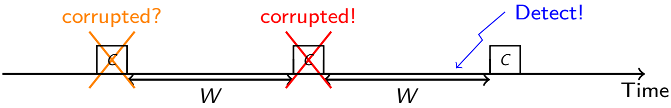


Keep multiple checkpoints?

Coping with Silent Errors

Silent error detected only when corrupted data is activated
e.g., soft faults in L1 cache, ALU, double bit flip.

Main problem: detection latency
Same approach?

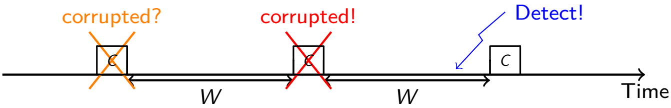


Keep multiple checkpoints?

Coping with Silent Errors

Silent error detected only when corrupted data is activated
e.g., soft faults in L1 cache, ALU, double bit flip.

Main problem: detection latency
Same approach?



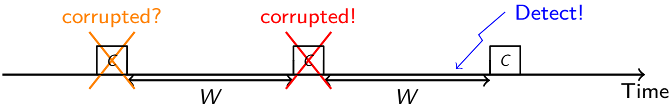
Keep multiple checkpoints?

Which checkpoint to recover from?

Coping with Silent Errors

Silent error detected only when corrupted data is activated
e.g., soft faults in L1 cache, ALU, double bit flip.

Main problem: detection latency
Same approach?



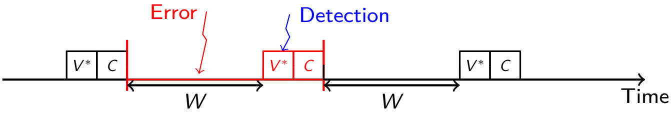
Keep multiple checkpoints?

Which checkpoint to recover from?

Need an active method to detect silent errors!

Coping with Silent Errors

Solution: coupling checkpointing with verification

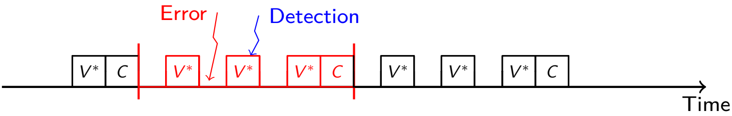


- Before each checkpoint, run some verification mechanism or error detection test
- Silent error, if any, is detected by verification
- Last checkpoint is always valid ☺

Problem solved! But can do better than that!

One step further

Perform several verifications before each checkpoint:

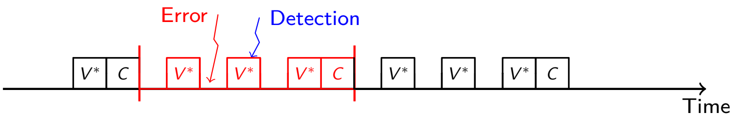


- Pro: silent error detected earlier in pattern ☺
- Con: additional overhead in error-free executions ☹

Seems good! ☺

One step further

Perform several verifications before each checkpoint:



- Pro: silent error detected earlier in pattern ☺
- Con: additional overhead in error-free executions ☹

Seems good! ☺

Wait... Verifications with 100% accuracy?

Partial verification

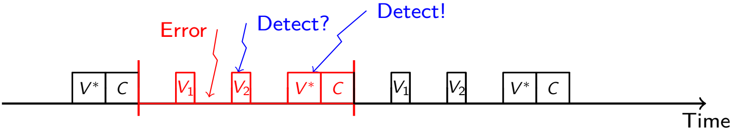
Guaranteed/perfect verifications (V^*) can be very expensive!
Partial verifications (V) are available for many HPC applications!

- Lower accuracy: recall $r = \frac{\#\text{detected errors}}{\#\text{total errors}} < 1$ ☹
- Much lower cost, i.e., $V < V^*$ ☺

Partial verification

Guaranteed/perfect verifications (V^*) can be very expensive!
Partial verifications (V) are available for many HPC applications!

- Lower accuracy: recall $r = \frac{\#\text{detected errors}}{\#\text{total errors}} < 1$ ☹
- Much lower cost, i.e., $V < V^*$ ☺

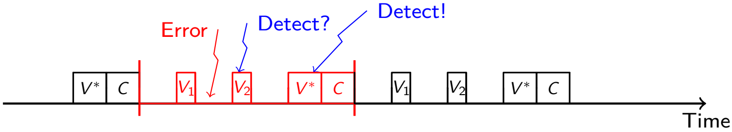


Ok! ☺

Partial verification

Guaranteed/perfect verifications (V^*) can be very expensive!
Partial verifications (V) are available for many HPC applications!

- Lower accuracy: recall $r = \frac{\#\text{detected errors}}{\#\text{total errors}} < 1$ ☹
- Much lower cost, i.e., $V < V^*$ ☺



Ok! ☺

Wait... Disk checkpoints are also expensive. Can we do better?

Two-level checkpointing

Two types of checkpoints

- Disk checkpoint: stable storage (slow but resilient)
- Memory checkpoint: local copy, (fast but lost on fail-stop)

Checkpoint only done after guaranteed verification.

Two-level checkpointing

Two types of checkpoints

- Disk checkpoint: stable storage (slow but resilient)
- Memory checkpoint: local copy, (fast but lost on fail-stop)

Checkpoint only done after guaranteed verification.

Two types of responses

- Fail-stop error ⇒ rollback to last disk checkpoint
- Silent errors ⇒ rollback to last memory checkpoint

Two-level checkpointing

Two types of checkpoints

- Disk checkpoint: stable storage (slow but resilient)
- Memory checkpoint: local copy, (fast but lost on fail-stop)

Checkpoint only done after guaranteed verification.

Two types of responses

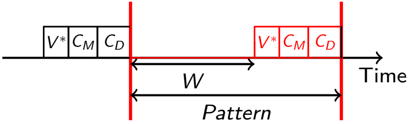
- Fail-stop error ⇒ rollback to last disk checkpoint
- Silent errors ⇒ rollback to last memory checkpoint

To do next:

- Combine everything into a single periodic pattern
- Minimize the expected execution time of the application

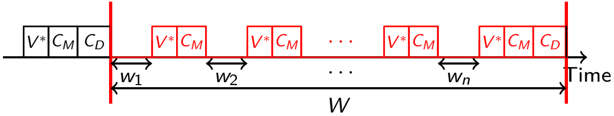
Resilience patterns (1/2)

Starting with base pattern



Pattern à la Young-Daly

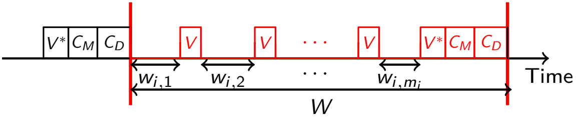
Adding verified memory checkpoints



Pattern with n segments

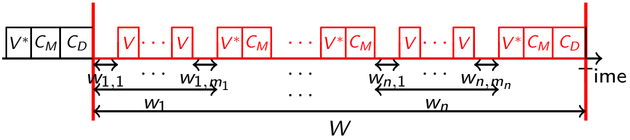
Resilience patterns (2/2)

Adding intermediate verifications between memory checkpoints



Segment w_i has m_i chunks

Putting everything together



Full pattern

Model (1/3)

Failure arrivals follow exponential law $Exp(\lambda)$, where $\lambda = 1/\mu$.

- Independant
- Memoryless

	Arrival rate	Probability of failure
fail-stop	λ_f	$p^f = 1 - e^{-\lambda_f w}$
silent	λ_s	$p^s = 1 - e^{-\lambda_s w}$

Same order.

$$\Leftrightarrow \lambda_f = \Theta(\lambda), \text{ and } \lambda_s = \Theta(\lambda)$$

where $\lambda = \lambda_f + \lambda_s = 1/\mu$ (platform MTBE)

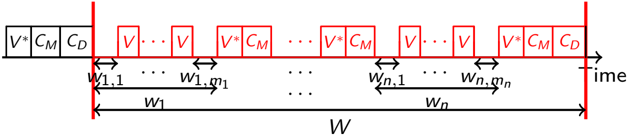
Model (2/3)

Two-level checkpointing.

- C_D cost of disk checkpointing (R_D for recovery)
- C_M cost of memory checkpointing (R_M for recovery)
- V cost of partial verification (with $r < 1$)
- V^* cost of guaranteed verification (with $r = 1$)

Model (3/3)

Finding optimal pattern

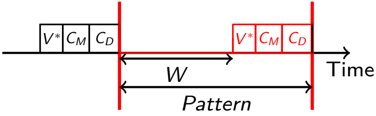


Total length	#Segments	#Chunks
W^*	n^*	m^*

Minimizing pattern overhead

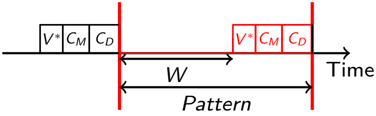
$$H(P) = \frac{\mathbb{E}(\mathrm{P})}{W} - 1$$

Derivation: how?



$$\mathbb{E}(\mathrm{P}) = p^f \left(\mathbb{E}(T^{\text{lost}}) + R_D + R_M + \mathbb{E}(\mathrm{P})\right)$$
$$+ (1 - p^f)\big(W + V^* + p^s(R_M + \mathbb{E}(\mathrm{P})) + (1 - p^s)(C_M + C_D)\big)$$

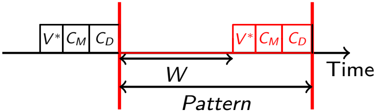
Derivation: how?



$$\mathbb{E}(\mathrm{P}) = p^f \left(\mathbb{E}(T^{\text{lost}}) + R_D + R_M + \mathbb{E}(\mathrm{P})\right) + (1-p^f)\big(W + V^* + p^s(R_M + \mathbb{E}(\mathrm{P})) + (1-p^s)(C_M + C_D)\big)$$

$$H(\mathrm{P}) = \frac{\mathbb{E}(\mathrm{P})}{W} - 1 = \frac{V^* + C_M + C_D}{W} + \left(\lambda_s + \frac{\lambda_f}{2}\right) W + \lambda_s(V^* + R_M) + \lambda_f(R_M + R_D) + O(\lambda^2 W^2)$$

Derivation: how?



$$\mathbb{E}(\mathrm{P}) = p^f \left(\mathbb{E}(T^{\text{lost}}) + R_D + R_M + \mathbb{E}(\mathrm{P})\right) + (1-p^f)\big(W + V^* + p^s(R_M + \mathbb{E}(\mathrm{P})) + (1-p^s)(C_M + C_D)\big)$$

$$H(\mathrm{P}) = \frac{\mathbb{E}(\mathrm{P})}{W} - 1 = \frac{V^* + C_M + C_D}{W} + \left(\lambda_s + \frac{\lambda_f}{2}\right) W + \lambda_s(V^* + R_M) + \lambda_f(R_M + R_D) + O(\lambda^2 W^2)$$

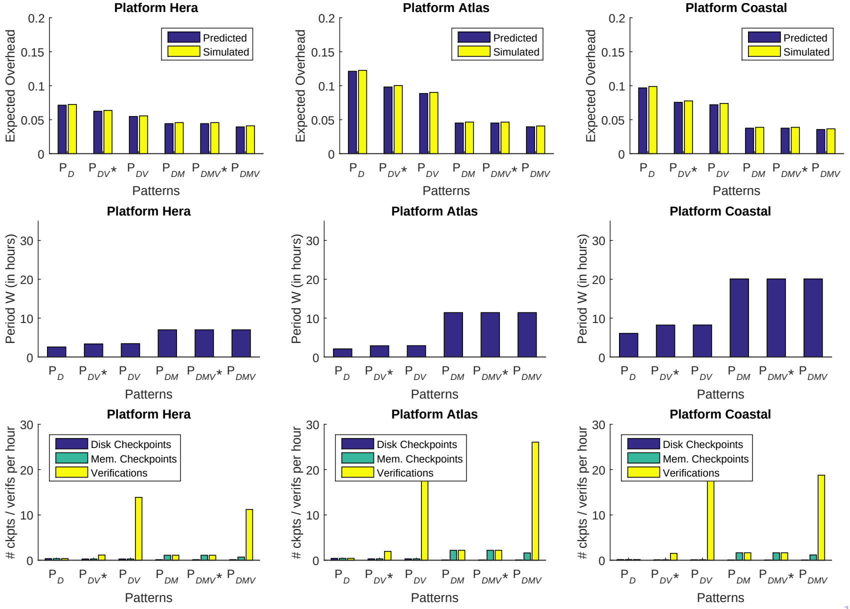
$$W^* = \sqrt{\frac{V^* + C_M + C_D}{\lambda_s + \frac{\lambda_f}{2}}}$$

$$H^*(\mathrm{P}) = 2\sqrt{\left(\lambda_s + \frac{\lambda_f}{2}\right)(V^* + C_M + C_D)} + O(\lambda)$$

Theorems

Pattern	W^*	n^*	m^*	$H^*(\mathrm{P})$
P_D	$\sqrt{\frac{V^*+C_M+C_D}{\lambda_s+\frac{\lambda_f}{2}}}$	–	–	$2\sqrt{\left(\lambda_s+\frac{\lambda_f}{2}\right)(V^*+C_M+C_D)}$
P_{DV^*}	$\sqrt{\frac{m^*V^*+C_M+C_D}{\frac{1}{2}\left(1+\frac{1}{m^*}\right)\lambda_s+\frac{\lambda_f}{2}}}$	–	$\sqrt{\frac{\lambda_s}{\lambda_s+\lambda_f}\cdot\frac{C_M+C_D}{V^*}}$	$\sqrt{2(\lambda_s+\lambda_f)C_M+C_D}+\sqrt{2\lambda_s V^*}$
P_{DV}	$\sqrt{\frac{(m^*-1)V+V^*+C_M+C_D}{\frac{1}{2}\left(1+\frac{2-r}{(m^*-2)r+2}\right)\lambda_s+\frac{\lambda_f}{2}}}$	–	$2-\frac{2}{r}+\sqrt{\frac{\lambda_s}{\lambda_s+\lambda_f}}$ $\times\sqrt{\frac{2-r}{r}\left(\frac{V^*+C_M+C_D}{V}-\frac{2-r}{r}\right)}$	$\sqrt{2(\lambda_s+\lambda_f)\left(V^*-\frac{2-r}{r}V+C_M+C_D\right)}$ $+\sqrt{2\lambda_s\frac{2-r}{r}V}$
P_{DM}	$\sqrt{\frac{n^*(V^*+C_M)+C_D}{\frac{\lambda_s}{n^*}+\frac{\lambda_f}{2}}}$	$\sqrt{\frac{2\lambda_s}{\lambda_f}\cdot\frac{C_D}{V^*+C_M}}$	–	$2\sqrt{\lambda_s(V^*+C_M)}+\sqrt{2\lambda_f C_D}$
P_{DMV^*}	$\sqrt{\frac{n^*m^*V^*+n^*C_M+C_D}{\frac{1}{2}\left(1+\frac{1}{m^*}\right)\frac{\lambda_s}{n^*}+\frac{\lambda_f}{2}}}$	$\sqrt{\frac{\lambda_s}{\lambda_f}\cdot\frac{C_D}{C_M}}$	$\sqrt{\frac{C_M}{V^*}}$	$\sqrt{2\lambda_f C_D}+\sqrt{2\lambda_s C_M}+\sqrt{2\lambda_s V^*}$
P_{DMV}	$\sqrt{\frac{n^*(m^*-1)V+n^*(V^*+C_M)+C_D}{\frac{1}{2}\left(1+\frac{2-r}{(m^*-2)r+2}\right)\frac{\lambda_s}{n^*}+\frac{\lambda_f}{2}}}$	$\sqrt{\frac{\lambda_s}{\lambda_f}\cdot\frac{C_D}{V^*-\frac{2-r}{r}V+C_M}}$	$2-\frac{2}{r}$ $+\sqrt{\frac{2-r}{r}\left(\frac{V^*+C_M}{V}-\frac{2-r}{r}\right)}$	$\sqrt{2\lambda_f C_D}+\sqrt{2\lambda_s\left(V^*-\frac{2-r}{r}V+C_M\right)}$ $+\sqrt{2\lambda_s\frac{2-r}{r}V}$

Experiments



Conclusion

Unified framework

- Error and application model
- Resilience patterns
- Optimal solutions

Next

- Multilevel fail-stop errors
- Replication vs checkpointing?

Thanks!

Methods for Detecting Silent Errors

General-purpose approaches

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

Application-specific approaches

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

Data-analytics approaches

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