Multi-level checkpointing and silent data corruption

Anne Benoit [2], Franck Cappello [1], Aurélien Cavelan [2], Sheng Di [1], Hongyang Sun [2], Yves Robert [2], Frédéric Vivien [2]

[1] Argonne National Laboratory [2] INRIA

October 6, 2016

Fail-stop errors

Characteristics

- Component failure (node, network, power, ...)
- Application fails and data is lost

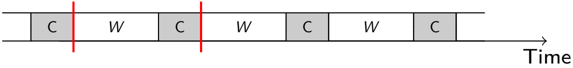
Fault rate proportional to number of components

- 2013: *Preprod.* Blue Waters requires repairs $\approx$ 4 hours [2, 1]
- 2014: Titan loses a node every $\approx$ 1.5 days [2, 3, 1]
- 2014: Blue Waters loses $\approx$ 2 nodes per day [1]

Coping with fail-stop errors

Instantaneous error detection

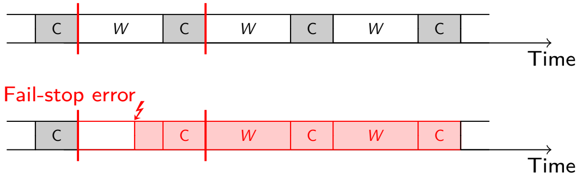
Standard approach: Periodic checkpoint, rollback, and recovery:



Coping with fail-stop errors

Instantaneous error detection

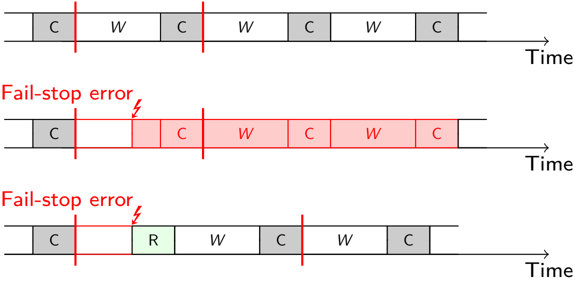
Standard approach: Periodic checkpoint, rollback, and recovery:



Coping with fail-stop errors

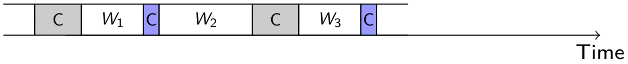
Instantaneous error detection

Standard approach: Periodic checkpoint, rollback, and recovery:



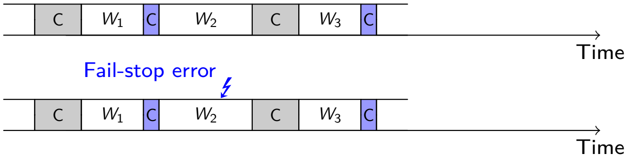
Multi-Level Checkpointing

- Different kinds of checkpoints: local disk storage, partner-copy, Reed-Solomon encoding technique, file system
- Different kinds of errors: node failure, router failure, etc.
- Each checkpoint has a cost and some resilience capabilities



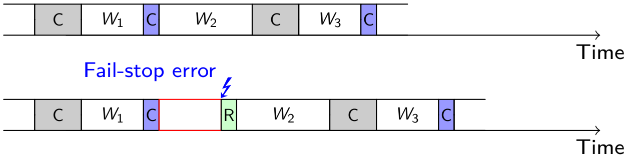
Multi-Level Checkpointing

- Different kinds of checkpoints: local disk storage, partner-copy, Reed-Solomon encoding technique, file system
- Different kinds of errors: node failure, router failure, etc.
- Each checkpoint has a cost and some resilience capabilities



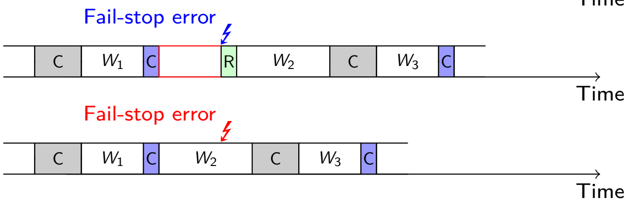
Multi-Level Checkpointing

- Different kinds of checkpoints: local disk storage, partner-copy, Reed-Solomon encoding technique, file system
- Different kinds of errors: node failure, router failure, etc.
- Each checkpoint has a cost and some resilience capabilities



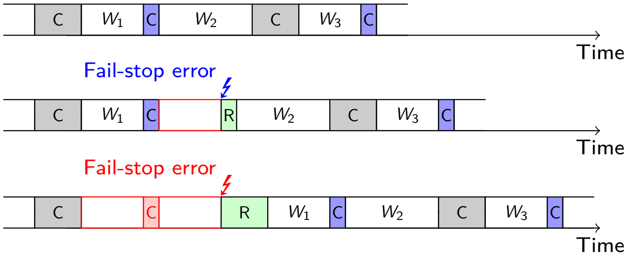
Multi-Level Checkpointing

- Different kinds of checkpoints: local disk storage, partner-copy, Reed-Solomon encoding technique, file system
- Different kinds of errors: node failure, router failure, etc.
- Each checkpoint has a cost and some resilience capabilities



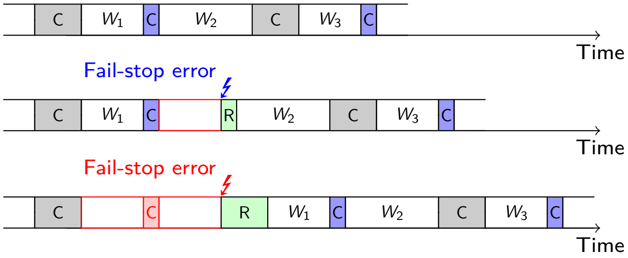
Multi-Level Checkpointing

- Different kinds of checkpoints: local disk storage, partner-copy, Reed-Solomon encoding technique, file system
- Different kinds of errors: node failure, router failure, etc.
- Each checkpoint has a cost and some resilience capabilities



Multi-Level Checkpointing

- Different kinds of checkpoints: local disk storage, partner-copy, Reed-Solomon encoding technique, file system
- Different kinds of errors: node failure, router failure, etc.
- Each checkpoint has a cost and some resilience capabilities



When should we checkpoint? Using which mechanism?

Two-level checkpointing: assumptions

Two types of faults

- Type-1: follow an exponential distribution of failure rate λ_1
- Type-2: follow an exponential distribution of failure rate λ_2

Two types of checkpoints

- Type-2 checkpoints take time C_2 (recovery R_2)
  Enables recovery from type-1 and type-2 faults

- Type-1 checkpoints take time C_1 (recovery R_1)
  Enables recovery from type-1 faults

Two-level checkpointing: assumptions

Two types of faults

- Type-1: follow an exponential distribution of failure rate λ_1
- Type-2: follow an exponential distribution of failure rate λ_2
  More dramatic faults

Two types of checkpoints

- Type-2 checkpoints take time C_2 (recovery R_2)
  Enables recovery from type-1 and type-2 faults
  More expensive checkpoints
- Type-1 checkpoints take time C_1 (recovery R_1)
  Enables recovery from type-1 faults Cheap checkpoints

Two-level checkpointing: assumptions

Two types of faults

- Type-1: follow an exponential distribution of failure rate λ_1
- Type-2: follow an exponential distribution of failure rate λ_2
  More dramatic faults

Two types of checkpoints

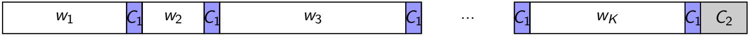
- Type-2 checkpoints take time C_2 (recovery R_2)
  Enables recovery from type-1 and type-2 faults
  More expensive checkpoints
- Type-1 checkpoints take time C_1 (recovery R_1)
  Enables recovery from type-1 faults Cheap checkpoints

Other assumptions

- Fault of type-i is followed by a *downtime* and a type-i recovery
- No faults during recoveries

Execution time of a pattern

- Pattern: work of some size W divided in K chunks



- Objective: overhead minimization

$$\text{OVERHEAD}(\text{PATTERN}(K, W, w_1, ..., w_K)) = \frac{\mathbb{E}(\text{PATTERN}(K, W, w_1, ..., w_K))}{W} - 1$$

- First property:
  Execution time is minimized when all chunks have same size

Unknown job length: optimal solution

- Chunks have size w_{opt} where:

$$N(w_{opt})\ln(N(w_{opt})) = \lambda L w_{opt}(e^{\lambda(w_{opt}+C_1)} - 1)$$

- There are K chunks in a pattern where:

$$\beta\lambda K w_{opt} e^{\lambda(w_{opt}+C_1)}(1 + L(e^{\lambda(w_{opt}+C_1)} - 1))^{K-1} = \alpha + \frac{\beta}{L}(1 + L(e^{\lambda(w_{opt}+C_1)} - 1))^{K}$$

- Missing notations
  $N(w) = 1 + L(e^{\lambda(w+C_1)} - 1)$, $L = \frac{\lambda_2}{\lambda}$, $\lambda = \lambda_1 + \lambda_2$,
  $\alpha = \mathcal{R}(e^{\lambda C_2} - 1) - \frac{\beta}{L}$, $\beta = \mathcal{R}(1 + L(e^{\lambda C_2} - 1))$,
  $\mathcal{R} = \frac{1+\lambda_1 R_1+\lambda_2 R_2}{\lambda} + D$
- Ugly implicit equations: solve them numerically!

Known job length: optimal solution

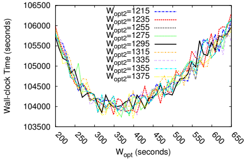
- Total size of job: $\mathcal{W}_{total}$
- Chunks have same w_{opt} size than previously
- There are p^* patterns where:

$$p^* = \frac{\mathcal{W}_{total} \ln(N(w_{opt}))}{\left(\mathbb{L}\left(\frac{\alpha L}{\beta e}\right) + 1\right) w_{opt}}$$

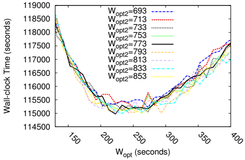
  with the same notations as previously
  and $\mathbb{L}(z) = x$ if $xe^x = z$.

- Ugly implicit equations: solve them numerically!

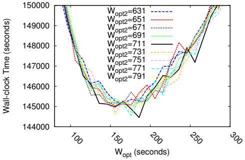
Assessment through simulations



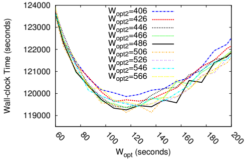
Case 1



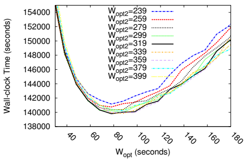
Case 2



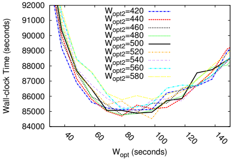
Case 3



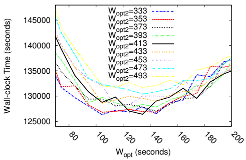
Case 4



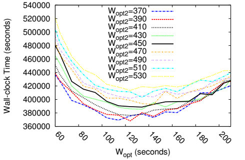
Case 5



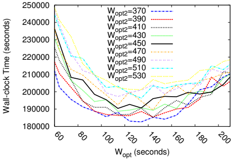
Case 6



Case 7



Case 8



Case 9

Conclusion so far

- We know how to use efficiently two-level checkpointing under fail-stop failures
- What about silent data corruption?

Second kind of errors: silent data corruption

Characteristics

- Bit flip (Disk, RAM, Cache, Bus, ...)
- Problems: detection latency, potentially wrong results

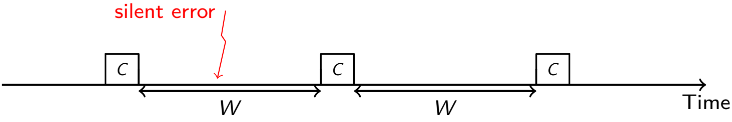
Cosmic rays do produce errors

- 2002: *Unprotected address bus* ASCI Q at Los Alamos National Laboratory could not run more than one hour [3]
- 2003: *No ECC* Virginia Tech 1,100 Apple Power Mac G5 supercomputer could not boot [3]
- 2010: *ECC protected* Jaguar saw 350 bit-flips/min [3]
- 2010: *ECC protected* Jaguar saw 1 double-bit error/day [3]
- 2014: Titan: *reported* $>$ 1 double-bit error per week [4]

Coping with Silent Errors

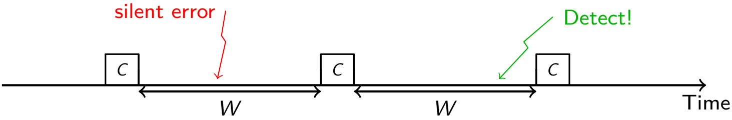
Main problem: detection latency

Question: can we follow the same approach?



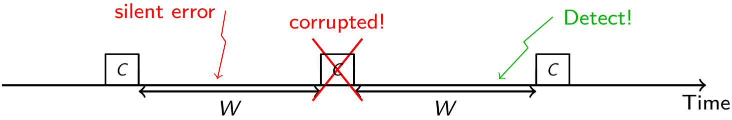
Coping with Silent Errors

Main problem: detection latency
Question: can we follow the same approach?



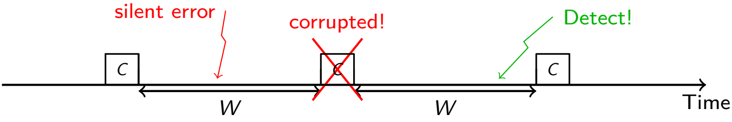
Coping with Silent Errors

Main problem: detection latency
Question: can we follow the same approach?



Coping with Silent Errors

Main problem: detection latency
Question: can we follow the same approach?

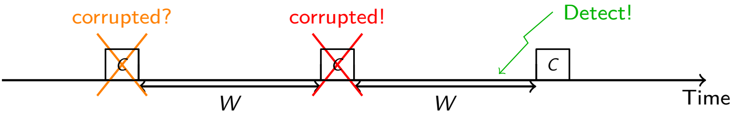


Keep multiple checkpoints?

Coping with Silent Errors

Main problem: detection latency

Question: can we follow the same approach?

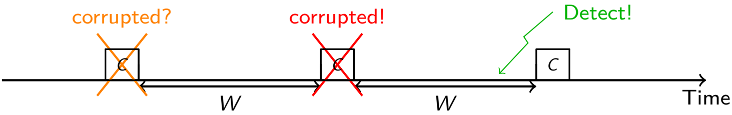


Keep multiple checkpoints?

Which checkpoint to recover from?

Coping with Silent Errors

Main problem: detection latency
Question: can we follow the same approach?



Keep multiple checkpoints?

Which checkpoint to recover from?

Need an active method to detect silent errors!

Existing Methods for Detecting Silent Errors

General-purpose approaches

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

Application-specific approaches

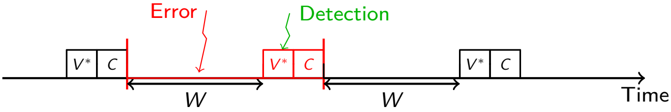
- Algorithm-based fault tolerance (ABFT): checksums in dense matrices Limited to one error detection and/or correction in practice [*Huang and Abraham 1984*]
- Partial differential equations (PDE): use lower-order scheme as verification mechanism [*Benson, Schmit and Schreiber 2014*]
- Generalized minimal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

Data-analytics approaches

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

Coping with Silent Errors

Solution: coupling checkpointing with verification

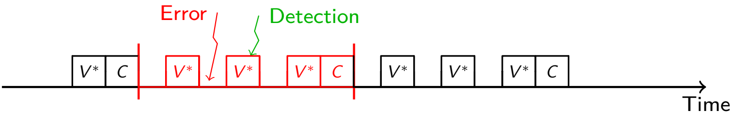


- Before each checkpoint, run some verification mechanism or error detection test
- Silent error, if any, is detected by verification
- Last checkpoint is always valid

Problem solved! But can do better than that!

One step further

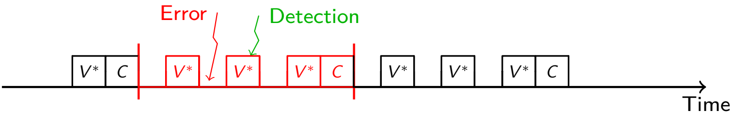
Perform several verifications before each checkpoint:



- Pro: silent error detected earlier in pattern
- Con: additional overhead in error-free executions
- Need to find the best trade-off

One step further

Perform several verifications before each checkpoint:

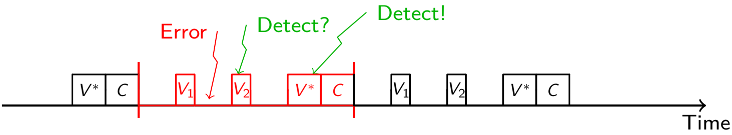


- Pro: silent error detected earlier in pattern
- Con: additional overhead in error-free executions
- Need to find the best trade-off
- Not all verification mechanisms have 100% accuracy!
  Should we use partial detectors? How?

Partial verification

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

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



The optimization problem

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 to errors

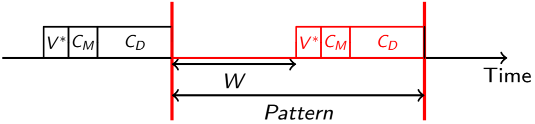
- Fail-stop error ⇒ rollback to last disk checkpoint
- Silent errors ⇒ rollback to last memory checkpoint

Goal:

- Combine everything into a single periodic pattern
- Minimize the overhead due to faults and to fault-tolerance

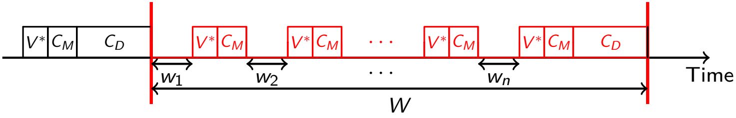
Resilience patterns (1/2)

Starting with base pattern



Pattern à la Young-Daly

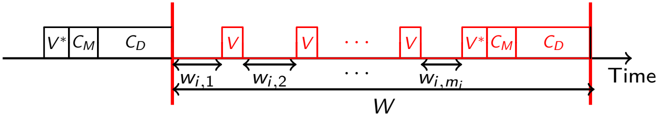
Adding verified memory checkpoints



Pattern with n segments

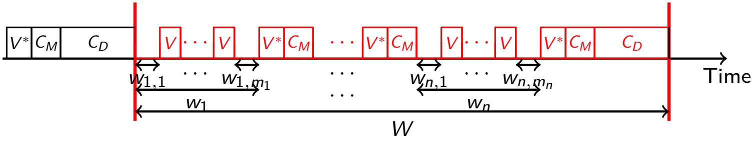
Resilience patterns (2/2)

Adding intermediate verifications between memory checkpoints



Segment w_i has m_i chunks

Putting everything together

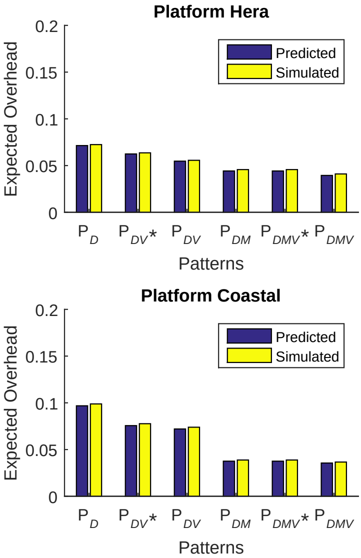


Full pattern

The optimal solution (first order approximation)

Pattern	W^*	n^*	m^*	OVERHEAD(PATTERN)
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}	$\sqrt{\frac{n^*(m^*-1)V+n^*(V^*+C_M)+C_D}{\frac{1}{2}\left(1+\frac{2-r}{(m^*-2)r+2}\right)\frac{\lambda_s}{n^*}+\frac{\lambda_f}{2}}}$	$\sqrt{\frac{\lambda_s}{\lambda_f}\cdot\frac{C_D}{V^*-\frac{2-r}{r}V+C_M}}$	$2-\frac{2}{r}$ $+\sqrt{\frac{2-r}{r}\left(\frac{V^*+C_M}{V}-\frac{2-r}{r}\right)}$	$\sqrt{2\lambda_f C_D}+\sqrt{2\lambda_s\left(V^*-\frac{2-r}{r}V+C_M\right)}$ $+\sqrt{2\lambda_s\frac{2-r}{r}V}$

Simulations



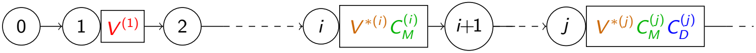
Conclusion so far

- We know how to use efficiently two-level checkpointing under fail-stop failures and silent data corruption with guaranteed verifications and partial verifications

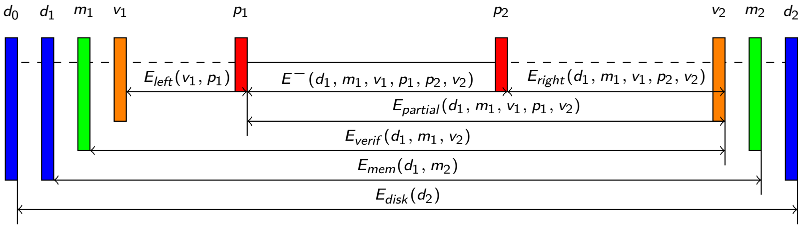
- Caveat: we assumed full freedom to place checkpoints and verifications (divisible load)
  Question: What about task graphs?

The optimization problem

- Application modeled as a linear task graph
- Checkpoints and verifications are performed in between tasks



- Question: when to take which checkpoint and verification in order to minimize the execution time?



- Optimal solution: $O(n^6)$ dynamic programming algorithm

Conclusion and perspectives

Pros

- Mix of silent and fail-stop errors
- Mix of partial and guaranteed verifications

Cons

- Results limited to 2 levels...
  ... but upcoming generalization for any number of levels!
- Exponential failure distribution

All details can be found in

- S. Di, Y. Robert, F. Vivien, and F. Cappello. Toward an Optimal Online Checkpoint Solution under a Two-Level HPC Checkpoint Model. *IEEE Transactions on Parallel and Distributed Systems*, 2016. To appear.
- A. Benoit, A. Cavelan, Y. Robert, and H. Sun. Optimal Resilience Patterns to Cope with Fail-Stop and Silent Errors. In *IPDPS'2016*, May 2016.
- A. Benoit, A. Cavelan, Y. Robert, and H. Sun. Two-Level Checkpointing and Verifications for Linear Task Graphs. In *The 17th IEEE International Workshop on Parallel and Distributed Scientific and Engineering Computing (PDSEC 2016)*, May 2016.

Bibliography I

P. Balaprakash, L. A. Bautista-Gomez, M. Bouguerra, S. M. Wild, F. Cappello, and P. D. Hovland.
Analysis of the tradeoffs between energy and run time for multilevel checkpointing.
In *5th International Workshop, PMBS 2014, New Orleans, LA, USA*, pages 249–263, 2014.

F. Cappello, G. Al, W. Gropp, S. Kale, B. Kramer, and M. Snir.
Toward exascale resilience: 2014 update.
Supercomput. Front. Innov.: Int. J., 1(1):5–28, Apr. 2014.

A. Geist.
How to kill a supercomputer: Dirty power, cosmic rays, and bad solder.
IEEE Spectrum, Feb. 2016.

Bibliography II

D. Tiwari, S. Gupta, J. H. Rogers, D. Maxwell, P. Rech, S. S. Vazhkudai, D. A. G. de Oliveira, D. Londo, N. DeBardeleben, P. O. A. Navaux, L. Carro, and A. S. Bland.
Understanding GPU errors on large-scale HPC systems and the implications for system design and operation.
In *21st IEEE International Symposium on High Performance Computer Architecture, HPCA 2015, Burlingame, CA, USA*, pages 331–342, 2015.

ANY QUESTIONS?