Which Verification for Soft Error Detection?

Leonardo Bautista-Gomez[1], Anne Benoit[2], Aurélien Cavelan[2], Saurabh K. Raina[3], Yves Robert[2,4] and Hongyang Sun[2]

1. Argonne National Laboratory, USA
2. ENS Lyon & INRIA, France
3. Jaypee Institute of Information Technology, India
4. University of Tennessee Knoxville, USA

hongyang.sun@ens-lyon.fr

JLESC Workshop
July 1, 2015, Barcelona

Computing at Exascale

Exascale platform:

- 10^5 or 10^6 nodes, each equipped 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	120 years
MTBF (platform of 10^6 nodes)	30 sec	5 mn	1 h

Computing at Exascale

Exascale platform:

- 10^5 or 10^6 nodes, each equipped 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

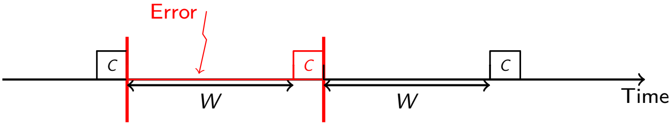
MTBF (individual node)	1 year	10 years	120 years
MTBF (platform of 10^6 nodes)	30 sec	5 mn	1 h

Need more reliable components!!

Need more resilient techniques!!!

General-purpose approach

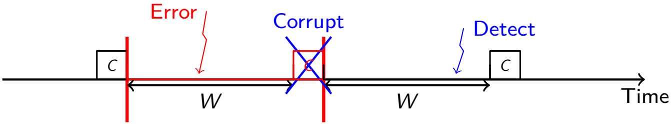
Periodic checkpoint, rollback and recovery:



- Fail-stop errors: instantaneous error detection, e.g., resource crash.
- Silent errors (aka silent data corruptions): e.g., soft faults in L1 cache, ALU, double bit flip.

General-purpose approach

Periodic checkpoint, rollback and recovery:



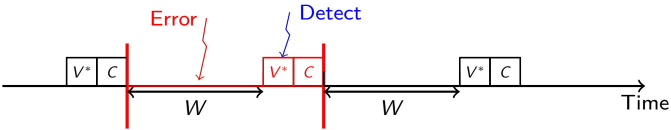
- Fail-stop errors: instantaneous error detection, e.g., resource crash.
- Silent errors (aka silent data corruptions): e.g., soft faults in L1 cache, ALU, double bit flip.

Silent error is detected only when corrupted data is activated, which could happen long after its occurrence.

Detection latency is problematic ⇒ risk of saving corrupted checkpoint!

Coping with silent errors

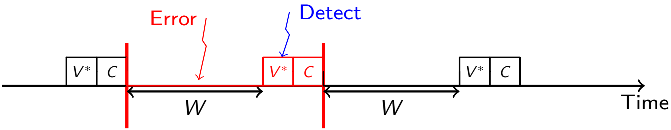
Couple checkpointing with verification:



- Before each checkpoint, run some verification mechanism (checksum, ECC, coherence tests, TMR, etc).
- Silent error is detected by verification ⇒ checkpoint always valid ☺

Coping with silent errors

Couple checkpointing with verification:



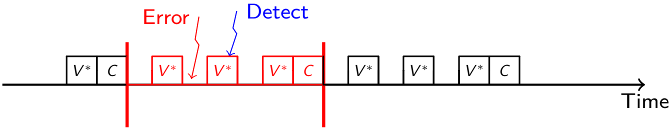
- Before each checkpoint, run some verification mechanism (checksum, ECC, coherence tests, TMR, etc).
- Silent error is detected by verification ⇒ checkpoint always valid ☺

Optimal period (Young/Daly):

	Fail-stop (classical)	Silent errors
Pattern	$T = W + C$	$S = W + V + C$
Optimal	$W^* = \sqrt{2C\mu}$	$W^* = \sqrt{(C+V)\mu}$

One step further

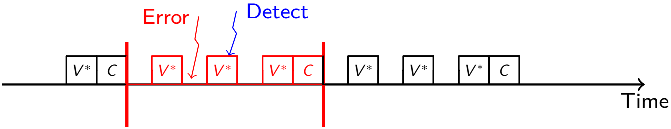
Perform several verifications before each checkpoint:



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

One step further

Perform several verifications before each checkpoint:



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

How many intermediate verifications to use and the positions?

Partial verification

Guaranteed/perfect verifications (V^*) can be very expensive!

Partial verifications (V) are available for many HPC applications!

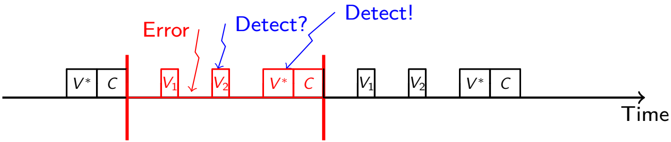
- Lower accuracy: recall $(r) = \frac{\#\text{detected errors}}{\#\text{total errors}} < 1$ ☹
- Much lower cost, i.e., $V < V^*$ ☺

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., $V < V^*$ ☺

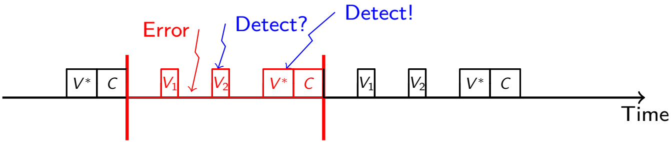


Partial verification

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., $V < V^*$ ☺



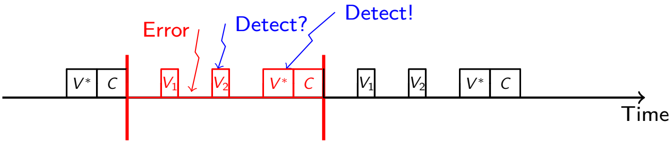
Which verification(s) to use? How many? Positions?

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., $V < V^*$ ☺



Which verification(s) to use? How many? Positions?

The terms "verification" and "detector" are used interchangeably.

Outline

Model and Objective

Divisible-load application

- Checkpoints and verifications can be inserted at arbitrary locations.

Silent errors

- Poisson process: arrival rate $\lambda = 1/\mu$, where μ is platform MTBF.
- Strike only computations; checkpointing, recovery, and verifications are protected.

Resilience parameters

- Cost of checkpointing C, cost of recovery R.
- k types of partial detectors and a perfect detector $\left(D^{(1)}, D^{(2)}, \dots, D^{(k)}, D^*\right)$.
  - $D^{(i)}$: cost $V^{(i)}$ and recall $r^{(i)} < 1$.
  - D^*: cost V^* and recall $r^* = 1$.

Model and Objective

Divisible-load application

- Checkpoints and verifications can be inserted at arbitrary locations.

Silent errors

- Poisson process: arrival rate $\lambda = 1/\mu$, where μ is platform MTBF.
- Strike only computations; checkpointing, recovery, and verifications are protected.

Resilience parameters

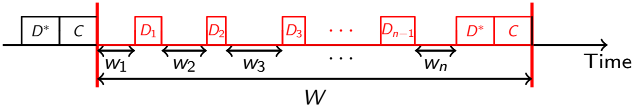
- Cost of checkpointing C, cost of recovery R.
- k types of partial detectors and a perfect detector $\left(D^{(1)}, D^{(2)}, \dots, D^{(k)}, D^*\right)$.
  - $D^{(i)}$: cost $V^{(i)}$ and recall $r^{(i)} < 1$.
  - D^*: cost V^* and recall $r^* = 1$.

Design an optimal periodic computing pattern that minimizes execution time (or makespan) of the application.

Pattern

Formally, a pattern $\textsc{Pattern}(W, n, \boldsymbol{\alpha}, \mathbf{D})$ is defined by

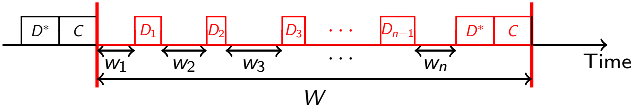
- W: pattern work length (or period);
- n: number of work segments;
- $\boldsymbol{\alpha} = [\alpha_1, \alpha_2, \ldots, \alpha_n]$: work fraction of each segment ($\alpha_i = w_i/W$ and $\sum_{i=1}^{n} \alpha_i = 1$);
- $\mathbf{D} = [D_1, D_2, \ldots, D_{n-1}, D^*]$: detectors used at the end of each segment ($D_i = D^{(j)}$ for some type j).



Pattern

Formally, a pattern $\textsc{Pattern}(W, n, \boldsymbol{\alpha}, \mathbf{D})$ is defined by

- W: pattern work length (or period);
- n: number of work segments;
- $\boldsymbol{\alpha} = [\alpha_1, \alpha_2, \ldots, \alpha_n]$: work fraction of each segment ($\alpha_i = w_i/W$ and $\sum_{i=1}^{n} \alpha_i = 1$);
- $\mathbf{D} = [D_1, D_2, \ldots, D_{n-1}, D^*]$: detectors used at the end of each segment ($D_i = D^{(j)}$ for some type j).



- Last detector is perfect to avoid saving corrupted checkpoints.
- The same detector type $D^{(j)}$ could be used at the end of several segments.

Outline

Summary of results

In a nutshell:

- We prove that finding the optimal pattern is NP-hard.
- We design an FPTAS (Fully Polynomial-Time Approximation Scheme) that gives a makespan within $(1 + \epsilon)$ times the optimal with running time polynomial in the input size and $1/\epsilon$.
- We show a simple Greedy algorithm works well in practice.

Summary of results

In a nutshell:

- We prove that finding the optimal pattern is NP-hard.
- We design an FPTAS (Fully Polynomial-Time Approximation Scheme) that gives a makespan within $(1+\epsilon)$ times the optimal with running time polynomial in the input size and $1/\epsilon$.
- We show a simple Greedy algorithm works well in practice.

Algorithm to determine a pattern $\text{PATTERN}(W, n, \alpha, \mathbf{D})$:

- Use FPTAS or Greedy (or even brute force for small instances) to find (optimal) number n of segments and set $\mathbf{D}$ of detectors.
- Arrange the $n-1$ partial detectors in any order.
- Compute $W^* = \sqrt{\frac{o_{\text{ff}}}{\lambda f_{\text{re}}}}$ and $\alpha_i^* = \frac{1}{U_n} \cdot \frac{1-g_{i-1}g_i}{(1+g_{i-1})(1+g_i)}$ for $1 \le i \le n$,

$$\text{where } o_{\text{ff}} = \sum_{i=1}^{n-1} V_i + V^* + C \text{ and } f_{\text{re}} = \frac{1}{2}\left(1 + \frac{1}{U_n}\right)$$

$$\text{with } g_i = 1 - r_i \text{ and } U_n = 1 + \sum_{i=1}^{n-1} \frac{1-g_i}{1+g_i}$$

Expected execution time of a pattern

Proposition

The expected time to execute a pattern $\textsc{Pattern}(W, n, \boldsymbol{\alpha}, \mathbf{D})$ *is*

$$\mathbb{E}(W) = W + \sum_{i=1}^{n-1} V_i + V^* + C + \lambda W(R + W\boldsymbol{\alpha}^T A\boldsymbol{\alpha} + \mathbf{d}^T\boldsymbol{\alpha}) + o(\lambda)$$

where A *is a symmetric matrix defined by* $A_{ij} = \frac{1}{2}\left(1 + \prod_{k=i}^{j-1} g_k\right)$ *for* $i \leq j$ *and* $\mathbf{d}$ *is a vector defined by* $\mathbf{d}_i = \sum_{j=i}^{n}\left(\prod_{k=i}^{j-1} g_k\right) V_i$ *for* $1 \leq i \leq n$.

- First-order approximation (as in Young/Daly's classic formula).

Expected execution time of a pattern

Proposition

The expected time to execute a pattern $\textsc{Pattern}(W, n, \boldsymbol{\alpha}, \mathbf{D})$ *is*

$$\mathbb{E}(W) = W + \sum_{i=1}^{n-1} V_i + V^* + C + \lambda W(R + W\boldsymbol{\alpha}^T A \boldsymbol{\alpha} + \mathbf{d}^T \boldsymbol{\alpha}) + o(\lambda)$$

where A *is a symmetric matrix defined by* $A_{ij} = \frac{1}{2}\left(1 + \prod_{k=i}^{j-1} g_k\right)$ *for* $i \leq j$ *and* $\mathbf{d}$ *is a vector defined by* $\mathbf{d}_i = \sum_{j=i}^{n}\left(\prod_{k=i}^{j-1} g_k\right) V_i$ *for* $1 \leq i \leq n$.

- First-order approximation (as in Young/Daly's classic formula).
- Matrix A is essential to analysis. For instance, when $n = 4$ we have:

$$A = \frac{1}{2}\begin{bmatrix} 2 & 1+g_1 & 1+g_1g_2 & 1+g_1g_2g_3 \\ 1+g_1 & 2 & 1+g_2 & 1+g_2g_3 \\ 1+g_1g_2 & 1+g_2 & 2 & 1+g_3 \\ 1+g_1g_2g_3 & 1+g_2g_3 & 1+g_3 & 2 \end{bmatrix}.$$

Minimizing makespan

For an application with total work W_{base}, the makespan is

$$\begin{aligned} W_{\text{final}} &\approx \frac{\mathbb{E}(W)}{W} \times W_{\text{base}} \\ &= W_{\text{base}} + H(W) \times W_{\text{base}}, \end{aligned}$$

where $H(W) = \frac{\mathbb{E}(W)}{W} - 1$ is the execution overhead.

For instance, if $W_{\text{base}} = 100$, $W_{\text{final}} = 120$, we have $H(W) = 20\%$.

Minimizing makespan

For an application with total work W_{base}, the makespan is

$$\begin{aligned} W_{\text{final}} &\approx \frac{\mathbb{E}(W)}{W} \times W_{\text{base}} \\ &= W_{\text{base}} + H(W) \times W_{\text{base}}, \end{aligned}$$

where $H(W) = \frac{\mathbb{E}(W)}{W} - 1$ is the execution overhead.

For instance, if $W_{\text{base}} = 100$, $W_{\text{final}} = 120$, we have $H(W) = 20\%$.

Minimizing makespan is equivalent to minimizing overhead!

$$H(W) = \frac{o_{\text{ff}}}{W} + \lambda f_{\text{re}} W + \lambda(R + \mathbf{d}^T \boldsymbol{\alpha}) + o(\lambda),$$

fault-free overhead: $$o_{\text{ff}} = \sum_{i=1}^{n-1} V_i + V^* + C,$$

re-execution fraction: $$f_{\text{re}} = \boldsymbol{\alpha}^T A \boldsymbol{\alpha}.$$

Optimal pattern length to minimize overhead

Proposition

The execution overhead of a pattern $\textsc{Pattern}(W, n, \alpha, \mathbf{D})$ *is minimized when its length is*

$$W^* = \sqrt{\frac{o_{ff}}{\lambda f_{re}}}.$$

The optimal overhead is

$$H(W^*) = 2\sqrt{\lambda o_{ff} f_{re}} + o(\sqrt{\lambda}).$$

Optimal pattern length to minimize overhead

Proposition

The execution overhead of a pattern $\text{PATTERN}(W, n, \alpha, \mathbf{D})$ *is minimized when its length is*

$$W^* = \sqrt{\frac{o_{ff}}{\lambda f_{re}}}.$$

The optimal overhead is

$$H(W^*) = 2\sqrt{\lambda o_{ff} f_{re}} + o(\sqrt{\lambda}).$$

- When the platform MTBF $\mu = 1/\lambda$ is large, $o(\sqrt{\lambda})$ is negligible.
- Minimizing overhead is reduced to minimizing the product $o_{\text{ff}} f_{\text{re}}$!
  - Tradeoff between fault-free overhead and fault-induced re-execution.

Optimal positions of verifications to minimize f_{re}

Theorem

The re-execution fraction f_{re} of a pattern $\textsc{Pattern}(W, n, \boldsymbol{\alpha}, \mathbf{D})$ *is minimized when* $\boldsymbol{\alpha} = \boldsymbol{\alpha}^*$, *where*

$$\alpha_k^* = \frac{1}{U_n} \times \frac{1 - g_{k-1}g_k}{(1+g_{k-1})(1+g_k)} \quad \text{for } 1 \leq k \leq n,$$

where $g_0 = g_n = 0$ *and* $U_n = 1 + \sum_{i=1}^{n-1} \frac{1-g_i}{1+g_i}$.
In this case, the optimal value of f_{re} *is*

$$f_{re}^* = \frac{1}{2}\left(1 + \frac{1}{U_n}\right).$$

Optimal positions of verifications to minimize f_{re}

Theorem

The re-execution fraction f_{re} of a pattern $\textsc{Pattern}(W, n, \boldsymbol{\alpha}, \mathbf{D})$ *is minimized when* $\boldsymbol{\alpha} = \boldsymbol{\alpha}^*$, *where*

$$\alpha_k^* = \frac{1}{U_n} \times \frac{1 - g_{k-1}g_k}{(1+g_{k-1})(1+g_k)} \quad \text{for } 1 \le k \le n,$$

where $g_0 = g_n = 0$ *and* $U_n = 1 + \sum_{i=1}^{n-1} \frac{1-g_i}{1+g_i}$.
In this case, the optimal value of f_{re} *is*

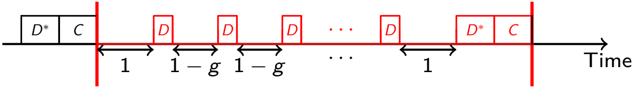
$$f_{re}^* = \frac{1}{2}\left(1 + \frac{1}{U_n}\right).$$

- Most technically involved result (lengthy proof of 3 pages!).
- Given a set of partial verifications, the minimal value of f_{re} does not depend upon their ordering within the pattern.

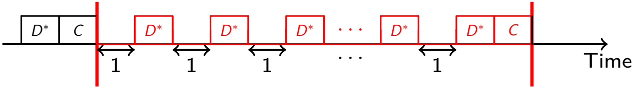
Two special cases

- When all verifications use the same partial detector (g), we get

$$\alpha_k^* = \begin{cases} \frac{1}{(n-2)(1-g)+2} & \text{for } k = 1 \text{ and } k = n \\ \frac{1-g}{(n-2)(1-g)+2} & \text{for } 2 \leq k \leq n-1 \end{cases}$$



- When all verifications use the perfect detector, we get equal-length segments, i.e., $\alpha_k^* = \frac{1}{n}$ for all $1 \leq k \leq n$.



Optimal number and set of detectors

It remains to determine optimal n and $\mathbf{D}$ of a pattern $\textsc{Pattern}(W, n, \boldsymbol{\alpha}, \mathbf{D})$.

Optimal number and set of detectors

It remains to determine optimal n and $\mathbf{D}$ of a pattern $\textsc{Pattern}(W, n, \boldsymbol{\alpha}, \mathbf{D})$.

Equivalent to the following optimization problem:

$$\text{Minimize} \quad f_{\text{re}} o_{\text{ff}} = \frac{V^* + C}{2}\left(1 + \frac{1}{1 + \sum_{j=1}^{k} m_j a^{(j)}}\right)\left(1 + \sum_{j=1}^{k} m_j b^{(j)}\right)$$

$$\text{subject to} \quad m_j \in \mathbb{N}_0 \quad \forall j = 1, 2, \ldots, k$$

accuracy: $a^{(j)} = \frac{1 - g^{(j)}}{1 + g^{(j)}}$ relative cost: $b^{(j)} = \frac{V^{(j)}}{V^* + C}$

accuracy-to-cost ratio: $\phi^{(j)} = \frac{a^{(j)}}{b^{(j)}}$

Optimal number and set of detectors

It remains to determine optimal n and $\mathbf{D}$ of a pattern $\textsc{Pattern}(W, n, \boldsymbol{\alpha}, \mathbf{D})$.

Equivalent to the following optimization problem:

$$\text{Minimize} \quad f_{\text{re}} o_{\text{ff}} = \frac{V^* + C}{2} \left(1 + \frac{1}{1 + \sum_{j=1}^{k} m_j a^{(j)}}\right) \left(1 + \sum_{j=1}^{k} m_j b^{(j)}\right)$$

$$\text{subject to} \quad m_j \in \mathbb{N}_0 \quad \forall j = 1, 2, \ldots, k$$

accuracy: $a^{(j)} = \frac{1 - g^{(j)}}{1 + g^{(j)}}$ relative cost: $b^{(j)} = \frac{V^{(j)}}{V^* + C}$

accuracy-to-cost ratio: $\phi^{(j)} = \frac{a^{(j)}}{b^{(j)}}$

NP-hard even when all detectors share the same accuracy-to-cost ratio (reduction from unbounded subset sum), but admits an FPTAS.

Greedy algorithm

Practically, a Greedy algorithm:

- Employs only the detector with highest accuracy-to-cost ratio $\phi^{\max} = \frac{a}{b}$.

$$\text{Optimal \#detectors: } m^* = -\frac{1}{a} + \sqrt{\frac{1}{a}\left(\frac{1}{b} - \frac{1}{a}\right)}$$

$$\text{Optimal overhead: } H^* = \sqrt{\frac{2(C + V^*)}{\mu}}\left(\sqrt{\frac{1}{\phi^{\max}}} + \sqrt{1 - \frac{1}{\phi^{\max}}}\right)$$

- Rounds up the optimal rational solution $\lceil m^* \rceil$.

Greedy algorithm

Practically, a Greedy algorithm:

- Employs only the detector with highest accuracy-to-cost ratio $\phi^{\max} = \frac{a}{b}$.

$$\text{Optimal \#detectors: } m^* = -\frac{1}{a} + \sqrt{\frac{1}{a}\left(\frac{1}{b} - \frac{1}{a}\right)}$$

$$\text{Optimal overhead: } H^* = \sqrt{\frac{2(C + V^*)}{\mu}}\left(\sqrt{\frac{1}{\phi^{\max}}} + \sqrt{1 - \frac{1}{\phi^{\max}}}\right)$$

- Rounds up the optimal rational solution $\lceil m^* \rceil$.

The Greedy algorithm has an approximation ratio $\sqrt{3/2} < 1.23$.

Outline

Simulation configuration

Exascale Platform:

- 10^5 computing nodes with individual MTBF of 100 years
  $\Rightarrow$ platform MTBF $\mu \approx 8.7$ hours.
- Checkpoints size of 300GB with throughput of 0.5GB/s
  $\Rightarrow C = 600s$.

Simulation configuration

Exascale Platform:

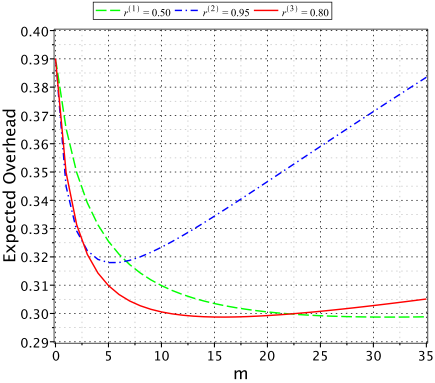
- 10^5 computing nodes with individual MTBF of 100 years $\Rightarrow$ platform MTBF $\mu \approx 8.7$ hours.
- Checkpoints size of 300GB with throughput of 0.5GB/s $\Rightarrow C = 600s$.

Realistic detectors (designed at ANL):

	cost	recall	ACR
Time series prediction $D^{(1)}$	$V^{(1)} = 3s$	$r^{(1)} = 0.5$	$\phi^{(1)} = 133$
Spatial interpolation $D^{(2)}$	$V^{(2)} = 30s$	$r^{(2)} = 0.95$	$\phi^{(2)} = 36$
Combination of the two $D^{(3)}$	$V^{(3)} = 6s$	$r^{(3)} = 0.8$	$\phi^{(3)} = 133$
Perfect detector D^*	$V^* = 600s$	$r^* = 1$	$\phi^* = 2$

Evaluation results

Using individual detector (Greedy algorithm)



Best partial detectors offer ∼9% improvement in overhead.
Saving ∼55 minutes for every 10 hours of computation!

Evaluation results

Mixing two detectors: depending on application or dataset, a detector's recall may vary, but its cost stays the same.

Realistic data again!

$r^{(1)} = [0.5, 0.9]$
$r^{(2)} = [0.75, 0.95]$
$r^{(3)} = [0.8, 0.99]$

$\phi^{(1)} = [133, 327]$
$\phi^{(2)} = [24, 36]$
$\phi^{(3)} = [133, 196]$

	m	overhead H	diff. from opt.
Scenario 1: $r^{(1)} = 0.51$, $r^{(3)} = 0.82$, $\phi^{(1)} \approx 137$, $\phi^{(3)} \approx 139$			
Optimal solution	(1, 15)	29.828%	0%
Greedy with $D^{(3)}$	(0, 16)	29.829%	0.001%
Scenario 2: $r^{(1)} = 0.58$, $r^{(3)} = 0.9$, $\phi^{(1)} \approx 163$, $\phi^{(3)} \approx 164$			
Optimal solution	(1, 14)	29.659%	0%
Greedy with $D^{(3)}$	(0, 15)	29.661%	0.002%
Scenario 3: $r^{(1)} = 0.64$, $r^{(3)} = 0.97$, $\phi^{(1)} \approx 188$, $\phi^{(3)} \approx 188$			
Optimal solution	(1, 13)	29.523%	0%
Greedy with $D^{(1)}$	(27, 0)	29.524%	0.001%
Greedy with $D^{(3)}$	(0, 14)	29.525%	0.002%

The Greedy algorithm works very well in this practical scenario!

Outline

Conclusion

A first comprehensive analysis of computing patterns with partial verifications to detect silent errors

- Theoretically: assess the complexity of the problem and propose efficient approximation schemes.
- Practically: present a Greedy algorithm and demonstrate its good performance with realistic detectors.

Conclusion

A first comprehensive analysis of computing patterns with partial verifications to detect silent errors

- Theoretically: assess the complexity of the problem and propose efficient approximation schemes.
- Practically: present a Greedy algorithm and demonstrate its good performance with realistic detectors.

Future directions

- Partial detectors with false positives/alarms

$$precision(p) = \frac{\#\text{true errors}}{\#\text{detected errors}} < 1.$$

- Errors in checkpointing, recovery, and verifications.
- Coexistence of fail-stop and silent errors.

Conclusion

A first comprehensive analysis of computing patterns with partial verifications to detect silent errors

- Theoretically: assess the complexity of the problem and propose efficient approximation schemes.
- Practically: present a Greedy algorithm and demonstrate its good performance with realistic detectors.

Future directions

- Partial detectors with false positives/alarms

$$precision(p) = \frac{\#\text{true errors}}{\#\text{detected errors}} < 1.$$

- Errors in checkpointing, recovery, and verifications.
- Coexistence of fail-stop and silent errors.

Research report available at https://hal.inria.fr/hal-01164445v1