Which verification for soft error detection?

Leonardo Bautista-Gomez\textsuperscript{1}, Anne Benoit\textsuperscript{2}, Aurélien Cavelan\textsuperscript{2}, Saurabh K. Raina\textsuperscript{3}, Yves Robert\textsuperscript{2,4} and Hongyang Sun\textsuperscript{2}

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

Anne.Benoit@ens-lyon.fr

Dagstuhl Seminar #15281: Algorithms and Scheduling Techniques to Manage Resilience and Power Consumption in Distributed Systems  
July 6, 2015, Schloss Dagstuhl, Germany
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) $\mu$

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

<table>
<thead>
<tr>
<th>MTBF (individual node)</th>
<th>1 year</th>
<th>10 years</th>
<th>120 years</th>
</tr>
</thead>
<tbody>
<tr>
<td>MTBF (platform of $10^6$ nodes)</td>
<td>30 sec</td>
<td>5 mn</td>
<td>1 h</td>
</tr>
</tbody>
</table>
Exascale platform:

- $10^5$ or $10^6$ nodes, each equipped with $10^2$ or $10^3$ cores
- Shorter Mean Time Between Failures (MTBF) $\mu$

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

<table>
<thead>
<tr>
<th>MTBF (individual node)</th>
<th>1 year</th>
<th>10 years</th>
<th>120 years</th>
</tr>
</thead>
<tbody>
<tr>
<td>MTBF (platform of $10^6$ nodes)</td>
<td>30 sec</td>
<td>5 mn</td>
<td>1 h</td>
</tr>
</tbody>
</table>

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

![Diagram showing periodic checkpoint, rollback, and recovery]

- 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!
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!
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 \(\Rightarrow\) 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):

<table>
<thead>
<tr>
<th>Pattern</th>
<th>Fail-stop (classical)</th>
<th>Silent errors</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$T = W + C$</td>
<td>$T = W + V^* + C$</td>
</tr>
<tr>
<td>Optimal</td>
<td>$W^* = \sqrt{2C \mu}$</td>
<td>$W^* = \sqrt{(C + V^*) \mu}$</td>
</tr>
</tbody>
</table>
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 😞
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., $V < V^*$ 😊
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?
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?
Model and objective

Silent errors
- Poisson process: arrival rate $\lambda = 1/\mu$, where $\mu$ 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 $(D^{(1)}, D^{(2)}, \ldots, D^{(k)}, D^*)$
  - $D^{(i)}$: 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
Formally, a pattern $\text{PATTERN}(W, n, \alpha, D)$ is defined by

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

- $W$: pattern work length (or period)
- $n$: number of work segments, of lengths $w_i$ (with $\sum_{i=1}^{n} w_i = W$)
- $\alpha = [\alpha_1, \alpha_2, \ldots, \alpha_n]$: work fraction of each segment ($\alpha_i = w_i/W$ and $\sum_{i=1}^{n} \alpha_i = 1$)
- $D = [D_1, D_2, \ldots, D_{n-1}, D^*]$: 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
In a nutshell:

- Given a pattern $\textsc{Pattern}(W, n, \alpha, D)$,
  - We show how to compute the \textit{expected execution time}
  - We are able to characterize its \textit{optimal length}
  - We can compute the \textit{optimal positions} of the partial verifications
Summary of results

In a nutshell:

- Given a pattern \( \text{PATTERN}(W, n, \alpha, D) \),
  - We show how to compute the expected execution time
  - We are able to characterize its optimal length
  - We can compute the optimal positions of the partial verifications

- However, 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 that works well in practice
Algorithm to determine a pattern $\text{Pattern}(W, n, \alpha, D)$:

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

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

with $g_i = 1 - r_i$ 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 \( \text{MODEL}(W, n, \alpha, D) \) is

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

where \( A \) is a symmetric matrix defined by

\[
A_{ij} = \frac{1}{2} \left( 1 + \prod_{k=i}^{j-1} g_k \right) \quad \text{for} \quad i \leq j \quad \text{and} \quad d \quad \text{is a vector defined by} \quad d_i = \sum_{j=i}^{n} \left( \prod_{k=i}^{j-1} g_k \right) V_i \quad \text{for} \quad 1 \leq i \leq n.
\]

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

\[
A = \frac{1}{2} \begin{bmatrix}
2 & 1 + g_1 & 1 + g_1 g_2 & 1 + g_1 g_2 g_3 \\
1 + g_1 & 2 & 1 + g_2 & 1 + g_2 g_3 \\
1 + g_1 g_2 & 1 + g_2 & 2 & 1 + g_3 \\
1 + g_1 g_2 g_3 & 1 + g_2 g_3 & 1 + g_3 & 2
\end{bmatrix}
\]
Minimizing makespan

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

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

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 \( W_{\text{base}} \), the makespan is

\[
W_{\text{final}} \approx \frac{\mathbb{E}(W)}{W} \times W_{\text{base}}
\]

\[
= W_{\text{base}} + H(W) \times W_{\text{base}},
\]

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!

\[
H(W) = \frac{O_{\text{off}}}{W} + \lambda f_{\text{re}} W + \lambda (R + d^T \alpha) + o(\lambda)
\]

fault-free overhead: \( O_{\text{off}} = \sum_{i=1}^{n-1} V_i + V^* + C \)

re-execution fraction: \( f_{\text{re}} = \alpha^T A \alpha \)
**Proposition**

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

$$W^* = \sqrt{\frac{\text{Off}}{\lambda_{\text{re}}}}.$$

The optimal overhead is

$$H(W^*) = 2\sqrt{\lambda_{\text{off}}\lambda_{\text{re}}} + o(\sqrt{\lambda}).$$
### Proposition

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

\[
W^* = \sqrt{\frac{\text{Off}}{\lambda f_{re}}}.
\]

The optimal overhead is

\[
H(W^*) = 2\sqrt{\lambda \text{Off} f_{re}} + o(\sqrt{\lambda}).
\]

- When the platform MTBF \( \mu = 1/\lambda \) is large, \( o(\sqrt{\lambda}) \) is negligible
- Minimizing overhead is reduced to minimizing the product \( \text{Off} f_{re} \)!
  - Tradeoff between fault-free overhead and fault-induced re-execution
The re-execution fraction $f_{re}$ of a pattern $\text{PATTERN}(W, n, \alpha, D)$ is minimized when $\alpha = \alpha^*$, 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 $\text{PATTERN}(W, n, \alpha, D)$ is minimized when $\alpha = \alpha^*$, where

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

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

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

- 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 ($r$), we get
  \[ \alpha_k^* = \begin{cases} 
  \frac{1}{n-2}r+2 & \text{for } k = 1 \text{ and } k = n \\
  \frac{r}{(n-2)r+2} & \text{for } 2 \leq k \leq n - 1 
  \end{cases} \]

- 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 $D$ of a pattern
$\text{Pattern}(W, n, \alpha, D)$.
Optimal number and set of detectors

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

Equivalent to the following optimization problem:

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

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

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 $D$ of a pattern $\text{Pattern}(W, n, \alpha, D)$.

Equivalent to the following optimization problem:

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

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

\[
\begin{align*}
\text{accuracy: } a(j) &= \frac{1 - g(j)}{1 + g(j)} \\
\text{relative cost: } b(j) &= \frac{V(j)}{V^* + C} \\
\text{accuracy-to-cost ratio: } \phi(j) &= \frac{a(j)}{b(j)}
\end{align*}
\]

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

- Employs only the detector with highest accuracy-to-cost ratio
  \[ \phi_{\text{max}} = \frac{a}{b} \]

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

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

- 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
  \[ \phi_{\text{max}} = \frac{a}{b} \]

Optimal number of detectors: \[ m^* = -\frac{1}{a} + \sqrt{\frac{1}{a} \left( \frac{1}{b} - \frac{1}{a} \right)} \]

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

- Rounds up the optimal rational solution \[ \lceil m^* \rceil \]

The greedy algorithm has an approximation ratio \[ \sqrt{3/2} < 1.23 \]
Simulation configuration

Exascale platform:

- $10^5$ computing nodes with individual MTBF of 100 years
  $\Rightarrow$ platform MTBF $\mu \approx 8.7$ hours

- Checkpoint sizes 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
- Checkpoint sizes of 300GB with throughput of 0.5GB/s
  $\Rightarrow$ $C = 600s$

Realistic detectors (designed at ANL):

<table>
<thead>
<tr>
<th>Detector Type</th>
<th>Cost $V$</th>
<th>Recall $r$</th>
<th>ACR $\phi$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Time series prediction $D^{(1)}$</td>
<td>$3s$</td>
<td>$0.5$</td>
<td>$133$</td>
</tr>
<tr>
<td>Spatial interpolation $D^{(2)}$</td>
<td>$30s$</td>
<td>$0.95$</td>
<td>$36$</td>
</tr>
<tr>
<td>Combination of the two $D^{(3)}$</td>
<td>$6s$</td>
<td>$0.8$</td>
<td>$133$</td>
</tr>
<tr>
<td>Perfect detector $D^*$</td>
<td>$600s$</td>
<td>$1$</td>
<td>$2$</td>
</tr>
</tbody>
</table>
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!

\[
\begin{align*}
r^{(1)} &= [0.5, 0.9] \\
r^{(2)} &= [0.75, 0.95] \\
r^{(3)} &= [0.8, 0.99] \\
\phi^{(1)} &= [133, 327] \\
\phi^{(2)} &= [24, 36] \\
\phi^{(3)} &= [133, 196]
\end{align*}
\]

<table>
<thead>
<tr>
<th>Scenario 1: ( r^{(1)} = 0.51, r^{(3)} = 0.82, \phi^{(1)} \approx 137, \phi^{(3)} \approx 139 )</th>
<th>( m )</th>
<th>overhead ( H )</th>
<th>diff. from opt.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Optimal solution</td>
<td>(1, 15)</td>
<td>29.828%</td>
<td>0%</td>
</tr>
<tr>
<td>Greedy with ( D^{(3)} )</td>
<td>(0, 16)</td>
<td>29.829%</td>
<td>0.001%</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Scenario 2: ( r^{(1)} = 0.58, r^{(3)} = 0.9, \phi^{(1)} \approx 163, \phi^{(3)} \approx 164 )</th>
<th>( m )</th>
<th>overhead ( H )</th>
<th>diff. from opt.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Optimal solution</td>
<td>(1, 14)</td>
<td>29.659%</td>
<td>0%</td>
</tr>
<tr>
<td>Greedy with ( D^{(3)} )</td>
<td>(0, 15)</td>
<td>29.661%</td>
<td>0.002%</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Scenario 3: ( r^{(1)} = 0.64, r^{(3)} = 0.97, \phi^{(1)} \approx 188, \phi^{(3)} \approx 188 )</th>
<th>( m )</th>
<th>overhead ( H )</th>
<th>diff. from opt.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Optimal solution</td>
<td>(1, 13)</td>
<td>29.523%</td>
<td>0%</td>
</tr>
<tr>
<td>Greedy with ( D^{(1)} )</td>
<td>(27, 0)</td>
<td>29.524%</td>
<td>0.001%</td>
</tr>
<tr>
<td>Greedy with ( D^{(3)} )</td>
<td>(0, 14)</td>
<td>29.525%</td>
<td>0.002%</td>
</tr>
</tbody>
</table>

The greedy algorithm works very well in this practical scenario!
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

Errors in checkpointing, recovery, and verifications

Coexistence of fail-stop and silent errors

Research report available at [https://hal.inria.fr/hal-01164445v1](https://hal.inria.fr/hal-01164445v1)
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

\[
\text{precision } p = \frac{\#\text{true errors}}{\#\text{detected errors}} < 1
\]

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

Research report available at https://hal.inria.fr/hal-01164445v1
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
  \[
  \text{precision } p = \frac{\#\text{true errors}}{\#\text{detected errors}} < 1
  \]
- Errors in checkpointing, recovery, and verifications
- Coexistence of fail-stop and silent errors

Research report available at [https://hal.inria.fr/hal-01164445v1](https://hal.inria.fr/hal-01164445v1)