Resilience Algorithms to Cope with Fail-Stop and Silent Errors

Hongyang Sun

ENS Lyon

hongyang.sun@ens-lyon.fr
http://perso.ens-lyon.fr/hongyang.sun/

HPC Days in Lyon
7 April, 2016
Joint work with

- Anne Benoit, Aurélien Cavelan, Yves Robert (ENS Lyon & Inria, France)
- Leonardo Bautista-Gomez (Argonne National Laboratory, USA)
- Saurabh K. Raina (Jaypee Institute of Information Technology, India)
Computing at Exascale

Exascale platform

- Larger node count: $10^5$ or $10^6$ nodes, each 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>100 years</th>
</tr>
</thead>
<tbody>
<tr>
<td>MTBF (platform of $10^6$ nodes)</td>
<td>30 secs</td>
<td>5 mins</td>
<td>50 mins</td>
</tr>
</tbody>
</table>

- Multiple failure sources: fail-stop error, silent data corruption, etc.

**Need more reliable components!**
**Need more scalable algorithms!**
**Need more resilient techniques!**
Fail-stop errors: e.g., resource crash, node failure

- Instantaneous error detection

Standard approach: periodic checkpointing, rollback and recovery

Well-known first-order approximation formula to compute optimal checkpointing interval [Young 1973, Daly 2006]:

\[ W^* = \sqrt{2\mu C} \]

\( \mu \): Platform MTBF
\( C \): Checkpointing time
Silent errors (or silent data corruptions): e.g., soft faults in L1 cache, ALU, double bit flip, due to cosmic radiation, packaging pollution, etc.

- Arbitrary detection latency

Same approach?

[Diagram showing the concept of silent errors and checkpoints over time]
Coping with Silent Errors

**Silent errors** (or silent data corruptions): e.g., soft faults in L1 cache, ALU, double bit flip, due to cosmic radiation, packaging pollution, etc.

- Arbitrary detection latency

**Same approach?**

![Diagram showing time intervals and silent error detection](image)
Silent errors (or silent data corruptions): e.g., soft faults in L1 cache, ALU, double bit flip, due to cosmic radiation, packaging pollution, etc.

- Arbitrary detection latency

Same approach?
Silent errors (or silent data corruptions): e.g., soft faults in L1 cache, ALU, double bit flip, due to cosmic radiation, packaging pollution, etc.

- Arbitrary detection latency

Same approach?

Keep multiple checkpoints?
Coping with Silent Errors

Silent errors (or silent data corruptions): e.g., soft faults in L1 cache, ALU, double bit flip, due to cosmic radiation, packaging pollution, etc.

- Arbitrary detection latency

Same approach?

Keep multiple checkpoints?
Coping with Silent Errors

**Silent errors** (or silent data corruptions): e.g., soft faults in L1 cache, ALU, double bit flip, due to cosmic radiation, packaging pollution, etc.

- Arbitrary detection latency

Same approach?

Keep multiple checkpoints?

Which checkpoint to recover from?
Coping with Silent Errors

Silent errors (or silent data corruptions): e.g., soft faults in L1 cache, ALU, double bit flip, due to cosmic radiation, packaging pollution, etc.

- Arbitrary detection latency

Same approach?

Keep multiple checkpoints?

Which checkpoint to recover from?

Need an active method to detect silent errors!
Coping with Silent Errors

Promising approach: coupling checkpointing with verification

- Before each checkpoint, run some verification mechanism or error detection test
- Silent error, if any, is detected by verification
- Need to maintain only one checkpoint, which is always valid 😊
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]
Methods for Detecting Silent Errors

General-purpose approaches

- Replication \[ \text{Fiala et al. 2012} \]
- Triple modular redundancy and voting \[ \text{Lyons and Vanderkulk 1962} \]

Application-specific approaches

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

Data-analytics approaches

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

Our focus is not about the design of silent error detectors. Instead, it is about how to make the best use of detectors (verifications) to design efficient resilience algorithms.

What is optimal checkpointing interval? Does intermediate verification help? What is the optimal verification position? How to cope with inaccurate detectors?
Outline

1. Coping with Silent Errors
   - Models
   - Analysis of several patterns

2. Coping with Fail-stop and Silent Errors

3. Conclusion and Future Work
Outline

1. Coping with Silent Errors
   - Models
     - Analysis of several patterns

2. Coping with Fail-stop and Silent Errors

3. Conclusion and Future Work
Failure arrivals follow exponential law \( \text{Exp}(\lambda) \), where \( \lambda = 1/\mu \).

\[-P(\lambda, w) = 1 - e^{\lambda w} \text{ (memoryless)}\]

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

A pattern has the following characteristics:
- End with a verified checkpoint (avoid saving corrupted checkpoints)
- May contain intermediate verifications (for better performance)
Models

Execution overhead

Suppose an application is divided into periodic patterns of work $W$. If the expected execution time of a pattern is $\mathbb{E}(W)$, then

\[
\text{Makespan} \approx \frac{\text{Total\_work}}{W} \cdot \mathbb{E}(W)
\]

\[
= (1 + H) \cdot \text{Total\_work}
\]

where

\[
H = \frac{\mathbb{E}(W)}{W} - 1
\]

denote the execution overhead of the pattern.

E.x. if $W = 100$, $\mathbb{E}(W) = 125$, then $H = 25\%$.

Proposition

For large applications, minimizing expected makespan is equivalent to minimizing the execution overhead of a pattern.
Outline

1. Coping with Silent Errors
   - Models
   - Analysis of several patterns

2. Coping with Fail-stop and Silent Errors

3. Conclusion and Future Work
Base Pattern $P_c$ (Revisiting Young/Daly)

**Proposition**

The optimal checkpointing interval $W^*$ and optimal execution overhead $H^*$ of the base pattern $P_c$ are

$$W^* = \sqrt{\frac{V^* + C}{\lambda}}$$

$$H^* = 2\sqrt{\lambda(V^* + C)} + O(\lambda)$$

<table>
<thead>
<tr>
<th>Pattern</th>
<th>Fail-stop errors</th>
<th>Silent errors</th>
</tr>
</thead>
<tbody>
<tr>
<td>Optimal $W^*$</td>
<td>$W + C$</td>
<td>$W + V^* + C$</td>
</tr>
<tr>
<td>Optimal $H^*$</td>
<td>$\sqrt{\frac{2C}{\lambda}}$</td>
<td>$2\sqrt{\lambda(V^* + C)}$</td>
</tr>
</tbody>
</table>
Pattern $P_{v^*c}$ with Intermediate Verifications

Can we do better by adding intermediate verifications in a pattern?

- Silent errors detected earlier in the pattern 😊
- Additional overhead in fault-free execution 😞
Pattern $P_{v^*c}$ with Intermediate Verifications

Can we do better by adding intermediate verifications in a pattern?

- Silent errors detected earlier in the pattern 😊
- Additional overhead in fault-free execution 😞

When is it better to use intermediate verifications?
What is the optimal checkpointing period?
How many verifications to use?
What are their positions?
Pattern $P_{v^*c}$ with Intermediate Verifications

The optimal $P_{v^*c}$ pattern has checkpointing interval $W^*$ and contains $n^*$ equi-spaced verifications:

$$n^* = \sqrt{C_{V^*}} \leftarrow \text{necessary condition: } C > V^*$$

$$W^* = \sqrt{n^* V^* + C_{\frac{1}{2}}} \leftarrow \text{base pattern}$$

$$H^* = \sqrt{2 V^* + \sqrt{2 V^* C_{\lambda}}} + O(\lambda) < 2 \sqrt{\frac{\lambda}{2}} (V^* + C_{\lambda}) + O(\lambda) \leftarrow \text{base pattern}$$

Practical no. of verifications must be an integer: $\max(1, \lfloor n^* \rfloor)$ or $\lceil n^* \rceil$
Pattern $P_{v^*c}$ with Intermediate Verifications

**Proposition**

The optimal $P_{v^*c}$ pattern has checkpointing interval $W^*$ and contains $n^*$ equi-spaced verifications:

\[ n^* = \sqrt{\frac{C}{V^*}} \quad \iff \text{necessary condition: } C > V^* \]

\[ W^* = \sqrt{\frac{n^* V^* + C}{\frac{1}{2} \left(1 + \frac{1}{n^*}\right) \lambda}} = \sqrt{\frac{2C}{\lambda}} > \sqrt{\frac{V^* + C}{\lambda}} \quad \iff \text{base pattern} \]

\[ H^* = \sqrt{2\lambda V^*} + \sqrt{2\lambda C} + O(\lambda) \]

\[ < 2\sqrt{\lambda(V^* + C)} + O(\lambda) \quad \iff \text{base pattern} \]
Proposition

The optimal $P_{v^*c}$ pattern has checkpointing interval $W^*$ and contains $n^*$ equi-spaced verifications:

\[
    n^* = \sqrt{\frac{C}{V^*}} \quad \Leftarrow \text{necessary condition: } C > V^*
\]

\[
    W^* = \sqrt{\frac{n^* V^* + C}{\frac{1}{2} (1 + \frac{1}{n^*}) \lambda}} = \sqrt{\frac{2C}{\lambda}} > \sqrt{\frac{V^* + C}{\lambda}} \quad \Leftarrow \text{base pattern}
\]

\[
    H^* = \sqrt{2\lambda V^* + \sqrt{2\lambda C} + O(\lambda)} < 2\sqrt{\lambda (V^* + C) + O(\lambda)} \quad \Leftarrow \text{base pattern}
\]

Practical no. of verifications must be an integer: $\max(1, \lfloor n^* \rfloor)$ or $\lceil n^* \rceil$
Observations

**Observation 1**

The expected time to execute a pattern of length $W$ is

$$
E(W) = \underbrace{W + o_{\text{off}}}_{\text{error-free time}} + \underbrace{\lambda W \cdot \left(f_{\text{re}} \cdot W + O(V^*) + R\right)}_{\text{expected re-execution time}} + O(\lambda)
$$

- $o_{\text{off}}$: overhead in a fault-free execution, i.e., $\sum$ resilience ops.
- $f_{\text{re}}$: fraction of re-executed work in case of faults.

**Observation 2**

The optimal pattern satisfies

$$
W^* = \sqrt{o_{\text{off}} \lambda f_{\text{re}}} H^* = 2 \sqrt{\lambda \cdot f_{\text{re}} o_{\text{off}}} + O(\lambda)
$$

Asymptotically, minimizing $H^*$ is equivalent to minimizing $f_{\text{re}} o_{\text{off}}$. 
Observations

Observation 1

The expected time to execute a pattern of length $W$ is

$$E(W) = W + o_{\text{off}} + \lambda W \cdot \left( f_{\text{re}} \cdot W + O(V^*) + R \right) + O(\lambda)$$

- $o_{\text{off}}$: overhead in a fault-free execution, i.e., $\sum$ resilience ops.
- $f_{\text{re}}$: fraction of re-executed work in case of faults.

Observation 2

The optimal pattern satisfies

$$W^* = \sqrt{\frac{o_{\text{off}}}{\lambda f_{\text{re}}}}$$
$$H^* = 2\sqrt{\lambda \cdot f_{\text{re}} o_{\text{off}}} + O(\lambda)$$
Observations

Observation 1

The expected time to execute a pattern of length $W$ is

$$E(W) = \underbrace{W + \text{off}}_{\text{error-free time}} + \underbrace{\lambda W}_{\text{expected #errors}} \cdot \left( f_{re} \cdot W + O(V^*) + R \right) + O(\lambda)$$

- $\text{off}$: overhead in a fault-free execution, i.e., $\sum$ resilience ops.
- $f_{re}$: fraction of re-executed work in case of faults.

Observation 2

The optimal pattern satisfies

$$W^* = \sqrt{\frac{\text{off}}{\lambda f_{re}}}$$

$$H^* = 2 \sqrt{\lambda \cdot f_{re} \text{off}} + O(\lambda)$$

Asymptotically, minimizing $H$ is equivalent to minimizing $f_{re} \text{off}$.
Some Observations

Example 1: Base pattern $P_c$

$$\mathbb{E}(W) = W + \left( V^* + C + \lambda W(\frac{1}{2} \cdot W + V^* + R) \right) + O(\lambda)$$

$$W^* = \sqrt{\frac{V^* + C}{\lambda}}$$

and $\mathbb{H}^* \approx 2\sqrt{\lambda(V^* + C)}$

Example 2: Pattern $P_{v^*c}$

$$\mathbb{E}(W) = W + \left( nV^* + C + \lambda W\left(\frac{1}{2} \cdot W + \frac{n + 1}{2} V^* + R\right) \right) + O(\lambda)$$

$$W^* = \sqrt{\frac{nV^* + C}{\frac{1}{2} \left(1 + \frac{1}{n}\right) \lambda}}$$

and $\mathbb{H}^* \approx 2\sqrt{\lambda \frac{1}{2} \left(nV^* + C\right) \left(1 + \frac{1}{n}\right)}$
Pattern $P_{vc}$ with Partial Verifications

**Guaranteed/perfect** verifications can be very expensive

For HPC applications, many silent error detectors are **partial**

- Lower cost 😊
- Lower accuracy 😞

\[
\text{cost } V \ll \text{cost } V^* \text{ of guaranteed verification}
\]

\[
\text{recall } r = \frac{\#\text{detected_errors}}{\#\text{total_errors}} < 1 \text{ (false negative)}
\]

\[
\text{precision } p = \frac{\#\text{true_errors}}{\#\text{detected_errors}} < 1 \text{ (false positive)}
\]
Can we do better by using partial verifications in a pattern?

- A partial verification may raise false alarms (with prob. $1 - p$)
- A partial verification may miss errors (with prob. $1 - r$)
- Last verification guaranteed to avoid saving invalid checkpoints
Pattern $P_{vc}$ with Partial Verifications

Can we do better by using partial verifications in a pattern?

- A partial verification may raise false alarms (with prob. $1 - p$)
- A partial verification may miss errors (with prob. $1 - r$)
- Last verification guaranteed to avoid saving invalid checkpoints

When is it better to use partial verifications?
What is the optimal checkpointing period?
How many partial verifications to use?
What are their positions?
The optimal pattern $P_{vc}$ does not use any partial verification with constant precision $p < 1$

In particular, the result holds if the precision satisfies $p = 1 - \Omega(\lambda^{1/2})$.

- Intuitively, an imprecise verification becomes another error source with error probability $1 - p$.
- With first-order approximation, probability of a silent error in the pattern is $1 - e^{\lambda W} \approx \lambda W = \Theta(\lambda^{1/2})$. 

The optimal pattern $P_{vc}$ does not use any partial verification with constant precision $p < 1$.

In particular, the result holds if the precision satisfies $p = 1 - \Omega(\lambda^{1/2})$.

- Intuitively, an imprecise verification becomes another error source with error probability $1 - p$.
- With first-order approximation, probability of a silent error in the pattern is $1 - e^{\lambda W} \approx \lambda W = \Theta(\lambda^{1/2})$.

Having a recall $r < 1$ is fine, because errors are rare and will eventually be detected by the final guaranteed verification.

**Tradeoff between recall and precision** ⇒ maximize precision
(e.g. $p > 0.999$ for $\lambda = 10^{-6}$)

We will assume $p = 1$ for subsequent analysis.
(1) Apply the $f_{re}o_{ff}$ analysis

**Proposition**

Suppose a pattern $P_{vc}$ has $n$ segments ($n - 1$ partial verifications and one guaranteed verification), and the $i$-th segment has $\alpha_i$ fraction of work. Then the pattern is characterized by

$$o_{ff} = (n - 1)V + V^* + C$$
$$f_{re} = \alpha^T A \alpha$$

where $\alpha = [\alpha_1, \alpha_2, \ldots, \alpha_n]^T$ and $A$ is a symmetric positive definite matrix defined by $A_{i,j} = \frac{1}{2} \left(1 + (1 - r)|i-j|\right)$ for $1 \leq i, j \leq n$
(2) Determine $\alpha$ to minimize $f_{re}$ (involved analysis)

**Proposition**

*The re-execution fraction $f_{re}$ of a pattern $P_{vc}$ with $n$ segments is minimized when $\alpha = \alpha^*$, where*

\[
\alpha_i^* = \begin{cases} 
\frac{1}{(n-2)r+2} & \text{for } i = 1, n \\
\frac{r}{(n-2)r+2} & \text{for } i = 2, 3, \ldots, n-1
\end{cases}
\]

*and the optimal value of $f_{re}$ is*

\[
f_{re}^* = \frac{1}{2} \left( 1 + \frac{2 - r}{(n-2)r + 2} \right)
\]

If all verifications are perfect ($r = 1$), we retrieve equal-length segments, i.e., $\alpha_i^* = \frac{1}{n}$ for all $1 \leq i \leq n$ and $f_{re}^* = \frac{1}{2} \left( 1 + \frac{1}{n} \right)$
Pattern $P_{vc}$ with Partial Verifications

(3) Minimize $f_{\text{reoff}} = \frac{1}{2} \left( 1 + \frac{2-r}{(n-2)r+2} \right) \left( (n-1)V + V^* + C \right)$

- accuracy $a = \frac{r}{2-r}$ and relative cost $b = \frac{V}{V^*+C}$
- accuracy-to-cost ratio $\phi = \frac{a}{b}$

Proposition

The optimal $P_{vc}$ pattern satisfies

$$n^* = 1 - \frac{1}{a} + \sqrt{\frac{1}{a} \left( \frac{1}{b} - \frac{1}{a} \right)} \quad \Leftarrow \text{necessary condition: } \phi > 2$$

$$W^* = \sqrt{\frac{2(V^*+C)}{\lambda}} \left( 1 - \frac{1}{\phi} \right) > \sqrt{\frac{2C}{\lambda}} \quad \Leftarrow \text{Pattern } P_{V^*_c}$$

$$H^* = \sqrt{2\lambda(V^*+C)} \left( \sqrt{\frac{1}{\phi}} + \sqrt{\frac{1}{\phi}} \right) + O(\lambda)$$

$$< \sqrt{2\lambda V^* + 2\lambda C} + O(\lambda) \quad \Leftarrow \text{Pattern } P_{V^*_c}$$
Assessing the benefit of partial verifications on realistic platform

- $10^5$ computing nodes with individual MTBF of 100 years
  ⇒ platform MTBF $\mu = 31536s \approx 8.7$ hours

- Checkpoint size of 300GB with throughput of 0.5GB/s
  ⇒ $C = 600s = 10$ mins, and $V^*$ in same order

- Partial verifications using lightweight detectors
  ⇒ $V$ typically tens of seconds, and $r \in [0.5, 0.95]$

  e.g., $C = 600$, $V^* = 300$, $V = 30$ and $p = 1$, $r = 0.8$

<table>
<thead>
<tr>
<th>Pattern $P_{vc}$</th>
<th>Pattern $P_{v^*c}$</th>
<th>Pattern $P_c$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$W^*$</td>
<td>7335s $\approx 2.04$ hours</td>
<td>7103s $\approx 1.97$ hours</td>
</tr>
<tr>
<td>$n^*$</td>
<td>6</td>
<td>2</td>
</tr>
<tr>
<td>$\alpha^*$</td>
<td>$\alpha_i =$  \begin{align*} 0.20, &amp; i = 1, 6 \ 0.15, &amp; i = 2..5 \end{align*}</td>
<td>[0.5, 0.5]</td>
</tr>
<tr>
<td>$H^*$</td>
<td>28.6%</td>
<td>33.3%</td>
</tr>
</tbody>
</table>
Can we do better by using multiple types of partial verifications?

\[ D^{(1)} = (V^{(1)}, r^{(1)}), \; D^{(2)} = (V^{(2)}, r^{(2)}), \; \ldots, \; D^{(k)} = (V^{(k)}, r^{(k)}) \]

The \( i \)-th partial verification has type \( j \), i.e., \( V_i = V^{(j)} \) for some \( 1 \leq j \leq k \)

Which verification is the optimal one to use?
What is the optimal combination of partial verifications?
**Proposition**

The optimal pattern $P_{vc}$ uses the partial verification with the *highest* accuracy-to-cost ratio

- Result is based on optimal rational solution ($n^*$)
- Overhead of rounded integer solution may no longer be optimal
Proposition

The optimal pattern $P_{vc}$ uses the partial verification with the highest accuracy-to-cost ratio.

- Result is based on optimal rational solution ($n^*$).
- Overhead of rounded integer solution may no longer be optimal.

What is the optimal integer solution?
Proposition

Finding the optimal $P_{vc}$ pattern with $k$ verification types is NP-complete, even when all verification types share the same accuracy-to-cost ratio, i.e., $\frac{a(j)}{b(j)} = \phi$ for all $1 \leq j \leq k$
Proposition

Finding the optimal $P_{vc}$ pattern with $k$ verification types is **NP-complete**, even when all verification types share the same accuracy-to-cost ratio, i.e., $\frac{a^{(j)}}{b^{(j)}} = \phi$ for all $1 \leq j \leq k$.

Approximation algorithms:

- **FPTAS (Fully Polynomial-Time Approximation Scheme)**
  - Overhead within $1 + \epsilon$ times the optimal with running time polynomial in the input size and $1/\epsilon$ for any $\epsilon > 0$.
  - The solution is independent of the ordering of the verifications.

- **Greedy algorithm**
  - Compute the optimal solution using the one detector with the highest accuracy-to-cost ratio, and then round up the solution.
  - This algorithm has approximation ratio $\sqrt{3}/2 < 1.23$.
Pattern with Multiple Partial Detectors

Performance evaluation on realistic 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
  $\Rightarrow C = 600s = 10$ mins, and $V^*$ in same order

- Several realistic partial detectors based on data-analytics approach

<table>
<thead>
<tr>
<th></th>
<th>cost</th>
<th>recall</th>
<th>ACR</th>
</tr>
</thead>
<tbody>
<tr>
<td>Time series prediction</td>
<td>$V^{(1)} = 3s$</td>
<td>$r^{(1)} = [0.5, 0.9]$</td>
<td>$\phi^{(1)} = [133, 327]$</td>
</tr>
<tr>
<td>Spatial interpolation</td>
<td>$V^{(2)} = 30s$</td>
<td>$r^{(2)} = [0.75, 0.95]$</td>
<td>$\phi^{(2)} = [24, 36]$</td>
</tr>
<tr>
<td>Combination of two</td>
<td>$V^{(3)} = 6s$</td>
<td>$r^{(3)} = [0.8, 0.99]$</td>
<td>$\phi^{(3)} = [133, 196]$</td>
</tr>
<tr>
<td>Perfect verification</td>
<td>$V^* = 600s$</td>
<td>$r^* = 1$</td>
<td>$\phi^* = 2$</td>
</tr>
</tbody>
</table>

A detector’s recall may vary depending on the application or dataset
Pattern with Multiple Partial Detectors

Using one type of verification ($r^{(1)} = 0.5$, $r^{(2)} = 0.95$, $r^{(3)} = 0.8$)

Best partial detectors offer $\sim 9\%$ improvement in overhead
Saving $\sim 55$ minutes for every 10 hours of computation!
Pattern with Multiple Partial Detectors

Using multiple types of verifications

<table>
<thead>
<tr>
<th>Scenario</th>
<th>$r^{(1)}$</th>
<th>$r^{(3)}$</th>
<th>$\phi^{(1)}$</th>
<th>$\phi^{(3)}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.51</td>
<td>0.82</td>
<td>137</td>
<td>139</td>
</tr>
<tr>
<td>Optimal solution</td>
<td>(1, 15)</td>
<td>29.828%</td>
<td>0%</td>
<td></td>
</tr>
<tr>
<td>Greedy with $V^{(3)}$</td>
<td>(0, 16)</td>
<td>29.829%</td>
<td>0.001%</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>0.58</td>
<td>0.9</td>
<td>163</td>
<td>164</td>
</tr>
<tr>
<td>Optimal solution</td>
<td>(1, 14)</td>
<td>29.659%</td>
<td>0%</td>
<td></td>
</tr>
<tr>
<td>Greedy with $V^{(3)}$</td>
<td>(0, 15)</td>
<td>29.661%</td>
<td>0.002%</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>0.64</td>
<td>0.97</td>
<td>188</td>
<td>188</td>
</tr>
<tr>
<td>Optimal solution</td>
<td>(1, 13)</td>
<td>29.523%</td>
<td>0%</td>
<td></td>
</tr>
<tr>
<td>Greedy with $V^{(1)}$</td>
<td>(27, 0)</td>
<td>29.524%</td>
<td>0.001%</td>
<td></td>
</tr>
<tr>
<td>Greedy with $V^{(3)}$</td>
<td>(0, 14)</td>
<td>29.525%</td>
<td>0.002%</td>
<td></td>
</tr>
</tbody>
</table>

The Greedy algorithm works very well in this practical setting!
Outline

1. Coping with Silent Errors
   - Models
   - Analysis of several patterns

2. Coping with Fail-stop and Silent Errors

3. Conclusion and Future Work
Coping with Fail-stop and Silent Errors

Fail-stop errors and silent errors **coexist** in large-scale platforms
A resilience pattern needs to cope with both error sources **simultaneously**
Coping with Fail-stop and Silent Errors

Fail-stop errors and silent errors coexist in large-scale platforms. A resilience pattern needs to cope with both error sources simultaneously.

Two-level checkpointing and verification framework

- **Fail-stop errors** \( (\lambda_f) \) are handled by disk checkpoints \( (C_D) \)
- **Silent errors** \( (\lambda_s) \) are handled by in-memory checkpoints \( (C_M) \) and verifications (guaranteed \( V^* \) or partial \( V \))

Framework enforcing two properties:

- A **guaranteed verification before each memory checkpoint**
  \[ \Rightarrow \text{Checkpoints always valid} \]
- A **memory checkpoint before each disk checkpoint**
  \[ \Rightarrow \text{Always recover from latest checkpoints} \]
**Two-level Base Pattern** $P_D$ (Revisiting Young/Daly Again)

The optimal checkpointing interval $W^*$ and optimal execution overhead $H^*$ of the two-level base pattern $P_D$ are

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

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

<table>
<thead>
<tr>
<th>Fail-stop errors</th>
<th>Silent errors</th>
<th>Both errors</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pattern</td>
<td>$W + C_D$</td>
<td>$W + V^* + C_M$</td>
</tr>
<tr>
<td>Optimal $W^*$</td>
<td>$\sqrt{\frac{2C_D}{\lambda_f}}$</td>
<td>$\sqrt{\frac{V^* + C_M}{\lambda_s}}$</td>
</tr>
<tr>
<td>Optimal $H^*$</td>
<td>$\sqrt{2\lambda_f C_D}$</td>
<td>$2\sqrt{\lambda_s (V^* + C_M)}$</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$2\sqrt{\left(\lambda_s + \frac{\lambda_f}{2}\right)(V^* + C_M + C_D)}$</td>
</tr>
</tbody>
</table>
Various Two-level Patterns

- Pattern $P_D$

- Pattern $P_{DM}$

- Pattern $P_{DV^*}$ or $P_{DV}$

- Pattern $P_{DMV^*}$ or $P_{DMV}$
Summary of Results

Parameters of various optimal patterns

- $W^*$: optimal pattern length
- $n^*$: optimal #memory checkpoints between two disk checkpoints
- $m^*$: optimal #verifications between two memory checkpoints

<table>
<thead>
<tr>
<th>Pattern</th>
<th>$W^*$</th>
<th>$n^*$</th>
<th>$m^*$</th>
<th>$H^*$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$P_D$</td>
<td>$\sqrt{\frac{V^*+C_M+C_D}{\lambda_s+\frac{\lambda_f}{2}}}$</td>
<td>-</td>
<td>-</td>
<td>$2\sqrt{\left(\lambda_s + \frac{\lambda_f}{2}\right) (V^* + C_M + C_D)}$</td>
</tr>
<tr>
<td>$P_{DV^*}$</td>
<td>$\sqrt{\frac{m^* V^<em>+C_M+C_D}{\frac{1}{2} \left(1+\frac{2}{m^</em>}\right) \lambda_s+\frac{\lambda_f}{2}}}$</td>
<td>-</td>
<td>$\sqrt{\frac{\lambda_s}{\lambda_s+\lambda_f} \cdot \frac{C_M+C_D}{V^*}}$</td>
<td>$\sqrt{2(\lambda_s + \lambda_f)C_M + C_D + \sqrt{2\lambda_s V^*}}$</td>
</tr>
<tr>
<td>$P_{DV}$</td>
<td>$\sqrt{\frac{(m^<em>-1)V+V^</em>+C_M+C_D}{\frac{1}{2} \left(1+\frac{2}{m^*}\right) \lambda_s+\frac{\lambda_f}{2}}}$</td>
<td>$2 - \frac{2}{r} + \sqrt{\frac{\lambda_s}{\lambda_s+\lambda_f}} \cdot \frac{C_D}{V^*+C_M}$</td>
<td>$\frac{2}{r} \left( V^* + C_M + C_D \right) - \frac{2}{r}$</td>
<td>$\sqrt{2(\lambda_s + \lambda_f) \left( V^* - \frac{2}{r} V + C_M + C_D \right)} + \sqrt{2\lambda_s \frac{2}{r} V}$</td>
</tr>
<tr>
<td>$P_{DM}$</td>
<td>$\sqrt{\frac{n^<em>(V^</em>+C_M)+C_D}{\frac{\lambda_s}{n^*}+\frac{\lambda_f}{2}}}$</td>
<td>$\sqrt{\frac{2\lambda_s}{\lambda_f} \cdot \frac{C_D}{V^*+C_M}}$</td>
<td>-</td>
<td>$2\sqrt{\lambda_s (V^* + C_M)} + \sqrt{2\lambda_f C_D}$</td>
</tr>
<tr>
<td>$P_{DMV^*}$</td>
<td>$\sqrt{\frac{n^* m^* V^<em>+n^</em> C_M+C_D}{\frac{1}{2} \left(1+\frac{1}{m^<em>}\right) \frac{\lambda_s}{n^</em>}+\frac{\lambda_f}{2}}}$</td>
<td>$\sqrt{\frac{\lambda_s}{\lambda_f} \cdot \frac{C_M}{C_M}}$</td>
<td>$\sqrt{\frac{C_M}{V^*}}$</td>
<td>$\sqrt{2\lambda_f C_D + \sqrt{2\lambda_s C_M} + \sqrt{2\lambda_s V^*}}$</td>
</tr>
<tr>
<td>$P_{DMV}$</td>
<td>$\sqrt{\frac{n^<em>(m^</em>-1)V+n^<em>(V^</em>+C_M)+C_D}{\frac{1}{2} \left(1+\frac{2-r}{(m^<em>-2)\frac{m^</em>}{r}+2}\right) \frac{\lambda_s}{n^*}+\frac{\lambda_f}{2}}}$</td>
<td>$\sqrt{\frac{\lambda_s}{\lambda_f} \cdot \frac{C_D}{V^*+\frac{2-r}{r} V+C_M}}$</td>
<td>$2 - \frac{2}{r}$</td>
<td>$\sqrt{2\lambda_f C_D} + \left( V^* - \frac{2-r}{r} V + C_M \right)$</td>
</tr>
</tbody>
</table>

$\lambda_s$: system load, $\lambda_f$: file load, $V^*$: virtual memory demand, $C_M$: memory cost, $C_D$: disk cost.
Parameters of four real platforms [Moody et al. 2010]

\[ V^* = C_M, \quad V = C_M/100 \quad \text{and} \quad r = 0.8 \]

<table>
<thead>
<tr>
<th>Platform</th>
<th>#nodes</th>
<th>( \lambda_f )</th>
<th>( \lambda_s )</th>
<th>( C_D )</th>
<th>( C_M )</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hera</td>
<td>256</td>
<td>9.46e-7</td>
<td>3.38e-6</td>
<td>300s</td>
<td>15.4s</td>
</tr>
<tr>
<td>Atlas</td>
<td>512</td>
<td>5.19e-7</td>
<td>7.78e-6</td>
<td>439s</td>
<td>9.1s</td>
</tr>
<tr>
<td>Coastal</td>
<td>1024</td>
<td>4.02e-7</td>
<td>2.01e-6</td>
<td>1051s</td>
<td>4.5s</td>
</tr>
<tr>
<td>Coastal SSD</td>
<td>1024</td>
<td>4.02e-7</td>
<td>2.01e-6</td>
<td>2500s</td>
<td>180.0s</td>
</tr>
</tbody>
</table>
A linear chain of $n$ tasks, and each task $T_i$ is characterized by a work $w_i$. Resilience operations (e.g., checkpoint, verification) possible only at the end of a task.

Which tasks to checkpoint (memory or disk) and which tasks to verify (guaranteed or partial) to minimize the expected makespan?

Optimal algorithm based on dynamic programming:

- Complexity $O(n^4)$ using only guaranteed verification
- Complexity $O(n^6)$ using also partial verification
Outline

1. Coping with Silent Errors
   - Models
   - Analysis of several patterns

2. Coping with Fail-stop and Silent Errors

3. Conclusion and Future Work
Summary

- First comprehensive analysis of computing patterns to cope with silent errors
- Two-level checkpointing and verification framework to cope with fail-stop and silent errors
- Optimal dynamic programming algorithms for linear task graph
- Performance evaluation based on parameters from real platforms

Future Work

- Analysis of multi-level/hierarchical checkpointing patterns
- Coping with failures in computational workflows modeled as directed acyclic graphs (DAGs)
References


