Revisiting the double checkpointing algorithm a.k.a the buddy algorithm

Jack Dongarra\textsuperscript{1}, Thomas Hérault\textsuperscript{1} and Yves Robert\textsuperscript{1,2}

1. University of Tennessee Knoxville, USA
2. Ecole Normale Supérieure de Lyon & INRIA, France

\{dongarra|herault\}@eecs.utk.edu, yves.robert@ens-lyon.fr
http://graal.ens-lyon.fr/~yrobert/

ICL Lunch Talk
Motivation

**buddy**

- **noun**
  - **forms**
  - **US** → *copain m*, *pote m*  
  - **esp US; [of Aids sufferer]** → *buddy m* (bénévoles accompagnant une personne atteinte du sida)

  - hi there, buddy! : salut, mon pote!

  - buddy movie or film: *film qui raconte l'histoire de deux amis*

- **compounds**
  - **buddy-buddy** → *adjective*
    - Paul and Mark are very buddy-buddy, Paul is very buddy-buddy with Mark:
      - Paul et Mark sont très copains or copains comme cochons
    - a buddy-buddy movie: *un film qui a pour héros deux amis*
Motivation

buddy  

1: a good friend that you are comfortable being around and sharing things with.
2: a condescending term used sarcastically to describe someone that you consider below yourself.

"I'm going to have some beers with my buddy tonight."

"Buddy just told me I can't park my car here."
Motivation

A nice word that men use in presenting some sort of emotional affection towards other men.

Man #1: I'm really scared.

Man #2: Everything is going to be okay, buddy.
Motivation

buddy/twitter/facebook

7 up, 4 down

a douchebag, asshole, or someone else that annoys you/gets on your nerves. Usually another driver on the road.

"Move it, buddy!"
Motivation

- Checkpoint transfer and storage
  ⇒ critical issues of rollback/recovery protocols

- Stable storage: high cost

- Distributed in-memory storage:
  - 😊 Much better scalability
  - 😞 Risk of non-recoverable failures
Outline

1. Double checkpointing algorithm
2. Analysis
3. Triple checkpointing algorithm
4. Experiments
Outline

1. Double checkpointing algorithm
2. Analysis
3. Triple checkpointing algorithm
4. Experiments
Main idea

- Store checkpoints in local memory ⇒ no centralized storage
- Replicate checkpoints ⇒ application survives single failure
- Still, risk of fatal failure in some (unlikely) scenarios
Double checkpoint algorithm

- Platform nodes partitioned into pairs
- Each node in a pair exchanges its checkpoint with its *buddy*
- Each node saves two checkpoints:
- one locally: storing its own data
- one remotely: receiving and storing its buddy’s data

*Two algorithms*
- blocking version by Zheng, Shi and Kalé
- non-blocking version by Ni, Meneses and Kalé
Non-blocking checkpoint algorithm

- Checkpoints taken periodically, with period $\mathcal{P} = \delta + \theta + \sigma$
- Phase 1, length $\delta$: local checkpoint, blocking mode. No work
- Phase 2, length $\theta$: remote checkpoint. Overhead $\phi$
- Phase 3, length $\sigma$: application at full speed $1$
Non-blocking checkpoint algorithm

Diagram:

Local checkpoint done

Remote checkpoint done

Period done

Node p

Node p'

δ
θ
P
σ

Work in failure-free period:

\[ W = (\theta - \phi) + \sigma = P - \delta - \phi \]
Cost of overlap

- Overlap computations and checkpoint file exchanges
- Large $\theta$
  $\Rightarrow$ more flexibility to hide cost of file exchange
  $\Rightarrow$ smaller overhead $\phi$

$\delta$ $\theta$ $\rho$ $\sigma$
Cost of overlap

- $\theta = \theta_{\text{min}}$: fastest communication, fully blocking $\Rightarrow \phi = \theta_{\text{min}}$
- $\theta = \theta_{\text{max}}$: full overlap with computation $\Rightarrow \phi = 0$
- Linear interpolation $\theta(\phi) = \theta_{\text{min}} + \alpha(\theta_{\text{min}} - \phi)$
  - $\phi = 0$ for $\theta = \theta_{\text{max}} = (1 + \alpha)\theta_{\text{min}}$
  - $\alpha$: rate of overhead decrease w.r.t. communication length

$\theta_{\text{min}}$: fastest communication, fully blocking
$\theta_{\text{max}}$: full overlap with computation
$\phi$: rate of overhead decrease
Assessing the risk

- After failure: downtime $D$ and recovery from buddy node
- Two checkpoint files lost, must be re-sent to faulty processor
  1. Checkpoint of faulty node, needed for recovery
     $\Rightarrow$ sent as fast as possible, in time $R = \theta_{\text{min}}$
  2. Checkpoint of buddy node, needed in case buddy fails later on
     $\Rightarrow$ ??

- Application at risk until complete reception of both messages
Checkpoint of buddy node

**Scenario DOUBLENBL**
- File sent at same speed as in regular mode, in time $\theta(\phi)$
- Overhead $\phi$
- Favors performance, at the price of higher risk

**Scenario DOUBLEBoF**
- File sent as fast as possible, in time $\theta_{\text{min}} = R$
- Overhead $R$
- Favors risk reduction, at the price of higher overhead
Outline

1. Double checkpointing algorithm
2. Analysis
3. Triple checkpointing algorithm
4. Experiments
Computing the waste

**Waste**

= fraction of time where nodes do not perform useful computations

- $T_{\text{base}}$: base time without any overhead due to resilience
- Time for fault-free execution $T_{\text{ff}}$
  - Period $P \Rightarrow W = P - \delta - \phi$ work units
  - $T_{\text{ff}} = \frac{P}{W} T_{\text{base}}$
  - $(1 - \frac{\delta + \phi}{P}) T_{\text{ff}} = T_{\text{base}}$
Computing the waste

- $T$ expectation of total execution time
  - $\rightarrow$ single application
  - $\rightarrow$ platform life (many jobs running concurrently)
- In average, failures occur every $M$ seconds
  - $\rightarrow$ platform MTBF $M = \mu_{\text{ind}}/p$
- For each failure, $\mathcal{F}$ seconds are lost:
  \[
  T = T_{\text{ff}} + \frac{T}{M} \mathcal{F}
  \]
  \[
  (1 - \frac{\mathcal{F}}{M})(1 - \frac{\delta + \phi}{\mathcal{P}}) T = T_{\text{base}}
  \]
Computing the waste

\[(1 - \text{WASTE}) \, T = T_{\text{base}}\]

\[
\text{WASTE} = 1 - \left(1 - \frac{F}{M}\right) \left(1 - \frac{\delta + \phi}{P}\right)
\]

Two sources of overhead:

\[
\text{WASTE}_{\text{ff}} = \frac{\delta + \phi}{P} : \text{checkpointing in a fault-free execution}
\]

\[
\text{WASTE}_{\text{fail}} = \frac{F}{M} : \text{failures striking during execution}
\]

\[
\text{WASTE} = \text{WASTE}_{\text{fail}} + \text{WASTE}_{\text{ff}} - \text{WASTE}_{\text{fail}} \, \text{WASTE}_{\text{ff}}
\]
Scenario DoubleNBL

\[ F_{\text{nbl}} = D + R + \frac{\delta}{P} R\mathcal{E}_1 + \frac{\theta}{P} R\mathcal{E}_2 + \frac{\sigma}{P} R\mathcal{E}_3 \]
Failure during third part of period

- No work during $D + R$
- Then re-execution of $W_{lost} = (\theta - \phi) + t_{lost}$
  - First $\theta$ seconds: overhead $\phi$ (receiving buddy checkpoint)
  - Then full speed
- $\mathbb{E}(t_{lost}) = \frac{\sigma}{2}$ (failures strike uniformly)

$\mathcal{RE}_3 = \theta + \frac{\sigma}{2}$

yves.robert@ens-lyon.fr
Waste minimization

Scenario **DOUBLENBL**

\[ F_{\text{nbl}} = D + R + \theta + \frac{P}{2} \]

\[ T \mathcal{O}_{\text{nbl}} = \sqrt{2(\delta + \phi)(M - R - D - \theta)} \]

Scenario **DOUBLEBOF**

\[ F_{\text{bof}} = F_{\text{nbl}} + R - \phi \]

\[ T \mathcal{O}_{\text{bof}} = \sqrt{2(\delta + \phi)(M - 2R - D - \theta + \phi)} \]

Not same \( \delta \) as in Young/Daly for coordinated checkpointing on global remote storage 😊
Waste minimization

Scenario **DoubleNBL** \( F_{\text{nbl}} = D + R + \theta + \frac{P}{2} \)

\[
T_O_{\text{nbl}} = \sqrt{2(\delta + \phi)(M - R - D - \theta)}
\]

Scenario **DoubleBoF** \( F_{\text{bof}} = F_{\text{nbl}} + R - \phi \)

\[
T_O_{\text{bof}} = \sqrt{2(\delta + \phi)(M - 2R - D - \theta + \phi)}
\]

Not same \( \delta \) as in Young/Daly for coordinated checkpointing on global remote storage 😊
Application at risk until complete reception of both messages:

- Risk = $D + R + \theta$ for DOUBLENBL
- Risk = $D + 2R$ for DOUBLEBoF

Analysis:

- Failures strike with uniform distribution over time
- $\lambda = \frac{1}{nM}$ instantaneous processor failure rate

**Success probability** $P_{\text{double}} = (1 - 2\lambda^2 T_{\text{Risk}})^{n/2}$
Consider a pair made of one processor and its buddy:

- Probability of first processor failing: $\lambda T$,
- Probability of one failure in the pair: $1 - (1 - \lambda T)^2 \approx 2\lambda T$
- Probability of second failure within risk period: $\lambda \text{Risk}$
- Probability of fatal failure in the pair: $(2\lambda T)(\lambda \text{Risk})$
- Probability of application fatal failure: $1 - (1 - 2\lambda^2 T \text{Risk})^{n/2}$

**Success probability**

$P_{\text{double}} = (1 - 2\lambda^2 T \text{Risk})^{n/2}$

$P_{\text{base}} = (1 - \lambda T_{\text{base}})^n$
Outline

1. Double checkpointing algorithm
2. Analysis
3. Triple checkpointing algorithm
4. Experiments
Principle

- Processors organized in triples
- Each processor has a preferred buddy and a secondary buddy
- Rotation of buddies
Waste in fault-free execution tends to zero

Application failure = three successive failures within a triple
⇒ Smaller risk even for large $\theta$

Only need non-blocking version TRIPLE
Memory requirement

- Copy-on-write for local checkpoint file
- Same memory usage as double checkpointing algorithm
Analysis

Waste

- $\text{WASTE}_{\text{fail}}$ same as for DoubleNBL
- $\text{WASTE}_{\text{ff}} = \frac{2\phi}{P}$ instead of $\text{WASTE}_{\text{ff}} = \frac{\delta + \phi}{P}$ for DoubleNBL

Risk

- Risk = $D + R + 2\theta$
- **Success probability** $\mathbb{P}_{\text{triple}} = (1 - 6\lambda^3 T\text{Risk}^2)^{n/3}$
Outline

1 Double checkpointing algorithm

2 Analysis

3 Triple checkpointing algorithm

4 Experiments
## Scenarios

<table>
<thead>
<tr>
<th>Scenario</th>
<th>$D$</th>
<th>$\delta$</th>
<th>$\phi$</th>
<th>$R$</th>
<th>$\alpha$</th>
<th>$n$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Base</td>
<td>0</td>
<td>2</td>
<td>$0 \leq \phi \leq 4$</td>
<td>4</td>
<td>10</td>
<td>$324 \times 32$</td>
</tr>
<tr>
<td>Exa</td>
<td>60</td>
<td>30</td>
<td>$0 \leq \phi \leq 60$</td>
<td>60</td>
<td>10</td>
<td>$10^6$</td>
</tr>
</tbody>
</table>
Waste for scenario *Base*

[Waste plots for DoubleBoF, DoubleNBL, and Triple]

Waste as a function of $\phi/R$ and $M$
Waste for scenario Base ($M = 7h$)
Success probability for scenario *Base*

Ratio $\text{DoubleNBL}/\text{DoubleBoF}$

Ratio $\text{DoubleBoF}/\text{Triple}$

Relative success probability function of $M$ and platform life $T$ ($\theta = (\alpha + 1)R$)
Waste for scenario *Exa*

**DoubleBoF**  
**DoubleNBL**  
**Triple**

Waste as a function of $\phi/R$ and $M$
Waste for scenario Exa ($M = 7h$)
Success probability for scenario \textit{Exa}

Ratio $\frac{\text{DoubleNBL}}{\text{DoubleBoF}}$  \hspace{1cm}  Ratio $\frac{\text{DoubleBoF}}{\text{Triple}}$

Relative success probability function of $M$ and platform life $T$ ($\theta = (\alpha + 1)R$)
Conclusion

Double checkpointing

- Revisiting algorithms by Zheng, Shi and Kalé and by Ni, Meneses and Kalé
- New version \textbf{DOUBLEBoF}: reduce risk duration, at the cost of increasing failure overhead
- New parameter $\alpha$ for transfer cost overlap
- Unified model for performance/risk bi-criteria assessment
Conclusion

**Triple checkpointing**

- Save checkpoint on two remote processes instead of one, without much more memory or storage requirements
- Excellent success probability, almost no failure-free overhead
- Assessment of performance and risk factors using unified mode
- Realistic scenarios conclude to superiority of TRIPLE
Future work

- Study real-life applications and propose refined values for $\alpha$ for a set of widely-used benchmarks
- Very small MTBF values on future exascale platforms
  $\Rightarrow$ combine distributed in-memory strategies with uncoordinated or hierarchical checkpointing protocols
Tribute to Buddy Knox

https://www.youtube.com/watch?v=yGrVNpFJbLI