Fault tolerance techniques for high-performance computing

Part 3

Anne Benoit

ENS Lyon

Anne.Benoit@ens-lyon.fr

http://graal.ens-lyon.fr/~abenoit

CR02 - 2015/2016
Outline

1. Probabilistic models

2. In-memory checkpointing
   - Double checkpointing algorithm
   - Analysis
   - Triple checkpointing algorithm
   - Experiments

3. Probabilistic models for advanced methods
   - Failure prediction
   - Replication

4. Forward-recovery techniques
   - ABFT for Linear Algebra applications
   - Composite approach: ABFT & Checkpointing

5. Conclusion
Outline

1. Probabilistic models
2. In-memory checkpointing
3. Probabilistic models for advanced methods
4. Forward-recovery techniques
5. Conclusion
Outline

1. Probabilistic models

2. In-memory checkpointing
   - Double checkpointing algorithm
   - Analysis
   - Triple checkpointing algorithm
   - Experiments

3. Probabilistic models for advanced methods

4. Forward-recovery techniques

5. Conclusion
Outline

1. Probabilistic models

2. In-memory checkpointing
   - Double checkpointing algorithm
   - Analysis
   - Triple checkpointing algorithm
   - Experiments

3. Probabilistic models for advanced methods

4. Forward-recovery techniques

5. Conclusion
Outline

1. Probabilistic models

2. In-memory checkpointing
   - Double checkpointing algorithm
   - Analysis
   - Triple checkpointing algorithm
   - Experiments

3. Probabilistic models for advanced methods

4. Forward-recovery techniques

5. Conclusion
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 $\mu$ seconds
  - $\rightarrow$ platform MTBF $\mu = \mu_{\text{ind}}/p$

- For each failure, $\mathcal{F}$ seconds are lost:

\[
T = T_{\text{ff}} + \frac{T}{\mu} \mathcal{F}
\]

\[
(1 - \frac{\mathcal{F}}{\mu})(1 - \frac{\delta + \phi}{P}) T = T_{\text{base}}
\]
Computing the waste

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

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

Two sources of overhead:

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

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

\[\text{WASTE} = \text{WASTE}_{\text{fail}} + \text{WASTE}_{\text{ff}} - \text{WASTE}_{\text{fail}} \text{WASTE}_{\text{ff}}\]
Time lost due to failures

Scenario **DOUBLENBL**

\[ \mathcal{F}_{nbl} = D + R + \frac{\delta}{P} \mathcal{RE}_1 + \frac{\theta}{P} \mathcal{RE}_2 + \frac{\sigma}{P} \mathcal{RE}_3 \]
No work during $D + R$

Then re-execution of $W_{\text{lost}} = (\theta - \phi) + t_{\text{lost}}$

- First $\theta$ seconds: overhead $\phi$ (receiving buddy checkpoint)
- Then full speed

$\mathbb{E}(t_{\text{lost}}) = \frac{\sigma}{2}$ (failures strike uniformly)

$$\mathcal{R}E_3 = \theta + \frac{\sigma}{2}$$
Waste minimization

Scenario DoubleNBL  \[ F_{\text{nbl}} = D + R + \theta + \frac{P}{2} \]
\[ T\theta_{\text{nbl}} = \sqrt{2(\delta + \phi)(\mu - R - D - \theta)} \]

Scenario DoubleBoF  \[ F_{\text{bof}} = F_{\text{nbl}} + R - \phi \]
\[ T\theta_{\text{bof}} = \sqrt{2(\delta + \phi)(\mu - 2R - D - \theta + \phi)} \]

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

Scenario DoubleNBL

\[ \mathcal{F}_{\text{nbl}} = D + R + \theta + \frac{P}{2} \]

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

Scenario DoubleBoF

\[ \mathcal{F}_{\text{bof}} = \mathcal{F}_{\text{nbl}} + R - \phi \]

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

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
- $\lambda = \frac{1}{n\mu}$ instantaneous processor failure rate

Success probability $P_{\text{double}} = (1 - 2\lambda^2 TRisk)^{n/2}$
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**

compare to

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

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

1. Probabilistic models
2. In-memory checkpointing
   - Double checkpointing algorithm
   - Analysis
   - Triple checkpointing algorithm
   - Experiments
3. Probabilistic models for advanced methods
4. Forward-recovery techniques
5. Conclusion
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
\[ \Rightarrow \text{Smaller risk even for large } \theta \]

Only need non-blocking version \text{T\textsc{riple}}
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 \( \text{DOUBLENBL} \)
- \( \text{WASTE}_{\text{ff}} = \frac{2\phi}{P} \) instead of \( \text{WASTE}_{\text{ff}} = \frac{\delta + \phi}{P} \) for \( \text{DOUBLENBL} \)

Risk

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

1. Probabilistic models

2. In-memory checkpointing
   - Double checkpointing algorithm
   - Analysis
   - Triple checkpointing algorithm
   - Experiments

3. Probabilistic models for advanced methods

4. Forward-recovery techniques

5. Conclusion
### 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><em>Base</em></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><em>Exa</em></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>

*Exa* corresponds to the Exa-Slim scenario.
Waste for scenario Base

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

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

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

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

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

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

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

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

1. Probabilistic models
2. In-memory checkpointing
3. Probabilistic models for advanced methods
   - Failure prediction
   - Replication
4. Forward-recovery techniques
5. Conclusion

Anne.Benoit@ens-lyon.fr CR02 Fault tolerance (3)
Outline

1. Probabilistic models
2. In-memory checkpointing
3. Probabilistic models for advanced methods
   - Failure prediction
   - Replication
4. Forward-recovery techniques
5. Conclusion
Framework

Predictor

- Exact prediction dates (at least $C$ seconds in advance)
- Recall $r$: fraction of faults that are predicted
- Precision $p$: fraction of fault predictions that are correct

Events

- true positive: predicted faults
- false positive: fault predictions that did not materialize as actual faults
- false negative: unpredicted faults
Fault rates

- \( \mu \): mean time between failures (MTBF)
- \( \mu_P \) mean time between predicted events (both true positive and false positive)
- \( \mu_{NP} \) mean time between unpredicted faults (false negative).
- \( \mu_e \): mean time between events (including three event types)

\[
\frac{1}{\mu_e} = \frac{1}{\mu_P} + \frac{1}{\mu_{NP}}
\]

\[
\begin{align*}
    r &= \frac{\text{True}_P}{\text{True}_P + \text{False}_N} \\
    p &= \frac{\text{True}_P}{\text{True}_P + \text{False}_P} \\

    \frac{1 - r}{\mu} &= \frac{1}{\mu_{NP}} \\
    \frac{r}{\mu} &= \frac{p}{\mu_P}
\end{align*}
\]
Example

- Predictor predicts six faults in time $t$
- Five actual faults. One fault not predicted
- $\mu = \frac{t}{5}$, $\mu_P = \frac{t}{6}$, and $\mu_{NP} = t$
- Recall $r = \frac{4}{5}$ (green arrows over red arrows)
- Precision $p = \frac{4}{6}$ (green arrows over blue arrows)
Algorithm

1. While no fault prediction is available:
   - checkpoints taken periodically with period $T$

2. When a fault is predicted at time $t$:
   - take a checkpoint ALAP (completion right at time $t$)
   - after the checkpoint, complete the execution of the period
Computing the waste

1. **Fault-free execution:** \( \text{WASTE}[FF] = \frac{C}{T} \)

2. **Unpredicted faults:** \( \frac{1}{\mu_{NP}} \left[ D + R + \frac{T}{2} \right] \)

\[
\text{WASTE}[\text{fail}] = \frac{1}{\mu} \left[ (1 - r) \frac{T}{2} + D + R + \frac{r}{p} C \right] \Rightarrow T_{opt} \approx \sqrt{\frac{2\mu C}{1 - r}}
\]
Computing the waste

3. Predictions: \[
\frac{1}{\mu P} \left[ p (C + D + R) + (1 - p)C \right]
\]

with actual fault (true positive)

no actual fault (false negative)

\[
W_{\text{Waste}}[\text{fail}] = \frac{1}{\mu} \left[ (1 - r) \frac{T}{2} + D + R + \frac{r}{p} C \right] \Rightarrow T_{opt} \approx \sqrt{\frac{2\mu C}{1 - r}}
\]
Computing the waste

3 Predictions: \[ \frac{1}{\mu p} \left[p(C + D + R) + (1 - p)C\right] \]

with actual fault (true positive)

\[ \text{WASTE}[\text{fail}] = \frac{1}{\mu} \left[ (1 - r) \frac{T}{2} + D + R + \frac{r}{p}C \right] \Rightarrow T_{opt} \approx \sqrt{\frac{2\mu C}{1 - r}} \]
Refinements

- Use different value $C_p$ for proactive checkpoints

- Avoid checkpointing too frequently for false negatives
  ⇒ Only trust predictions with some fixed probability $q$
  ⇒ Ignore predictions with probability $1 - q$

Conclusion: trust predictor always or never ($q = 0$ or $q = 1$)

- Trust prediction depending upon position in current period
  ⇒ Increase $q$ when progressing
  ⇒ Break-even point $\frac{C_p}{p}$
With prediction windows

(Regular mode)

\[ C \quad T_R-C \quad C \quad T_R-C \quad T_{lost} \quad T_R-C \quad C \]

(Prediction without failure)

\[ C \quad T_R-C \quad \underbrace{W_{reg}}_{\text{Regular mode}} \quad C_p \quad C_p \quad C_p \quad C_p \quad \overbrace{T_P-C_p}^{\text{Proactive mode}} \quad C \]

(Prediction with failure)

\[ C \quad T_R-C \quad \underbrace{W_{reg}}_{\text{Regular mode}} \quad C_p \quad C_p \quad C_p \quad C_p \quad \overbrace{D}^{\text{Error}} \quad \overbrace{T_P-C_p}^{\text{Proactive mode}} \quad C \]

Error

Gets too complicated! 😞
Outline

1. Probabilistic models
2. In-memory checkpointing
3. Probabilistic models for advanced methods
   - Failure prediction
   - Replication
4. Forward-recovery techniques
5. Conclusion
Replication

- Systematic replication: efficiency < 50%
- Can replication + checkpointing be more efficient than checkpointing alone?
- Study by Ferreira et al. [SC’2011]: yes
Model by Ferreira et al. [SC’ 2011]

- Parallel application comprising $N$ processes
- Platform with $p_{total} = 2N$ processors
- Each process replicated $\rightarrow$ $N$ replica-groups
- When a replica is hit by a failure, it is not restarted
- Application fails when both replicas in one replica-group have been hit by failures
Example

Pair_1

Pair_2

Pair_3

Pair_4

Time

Fault tolerance (3)
The birthday problem

Classical formulation
What is the probability, in a set of $m$ people, that two of them have same birthday?

Relevant formulation
What is the average number of people required to find a pair with same birthday?

\[
Birthday(N) = 1 + \int_0^{+\infty} e^{-x}(1 + x/N)^{N-1} dx
\]

The analogy

Two people with same birthday
\[\equiv\]
Two failures hitting same replica-group
Differences with birthday problem

- $N$ processes; each replicated twice
- Uniform distribution of failures
- First failure: each replica-group has probability $1/N$ to be hit
- Second failure
Differences with birthday problem

- $N$ processes; each replicated twice
- Uniform distribution of failures
- First failure: each replica-group has probability $1/N$ to be hit
- Second failure
Differences with birthday problem

- $N$ processes; each replicated twice
- Uniform distribution of failures
- First failure: each replica-group has probability $1/N$ to be hit
- Second failure: can failed PE be hit?
Differences with birthday problem

- \( N \) processes; each replicated twice
- Uniform distribution of failures
- First failure: each replica-group has probability \( 1/N \) to be hit
- Second failure cannot hit failed PE
  - Failure uniformly distributed over \( 2N - 1 \) PEs
  - Probability that replica-group \( i \) is hit by failure: \( 1/(2N - 1) \)
  - Probability that replica-group \( \neq i \) is hit by failure: \( 2/(2N - 1) \)
  - Failure not uniformly distributed over replica-groups: this is not the birthday problem
Differences with birthday problem

- $N$ processes; each replicated twice
- Uniform distribution of failures
- First failure: each replica-group has probability $1/N$ to be hit
- Second failure cannot hit failed PE
  - Failure uniformly distributed over $2N - 1$ PEs
  - Probability that replica-group $i$ is hit by failure: $1/(2N - 1)$
  - Probability that replica-group $\neq i$ is hit by failure: $2/(2N - 1)$
  - Failure not uniformly distributed over replica-groups: this is not the birthday problem
Differences with birthday problem

- \( N \) processes; each replicated twice
- Uniform distribution of failures
- First failure: each replica-group has probability \( 1/N \) to be hit
- Second failure cannot hit failed PE
  - Failure uniformly distributed over \( 2N - 1 \) PEs
  - Probability that replica-group \( i \) is hit by failure: \( 1/(2N - 1) \)
  - Probability that replica-group \( \neq i \) is hit by failure: \( 2/(2N - 1) \)
- Failure not uniformly distributed over replica-groups: this is not the birthday problem
Differences with birthday problem

- \( N \) processes; each replicated twice
- Uniform distribution of failures
- First failure: each replica-group has probability \( 1/N \) to be hit
- Second failure **cannot** hit failed PE
  - Failure uniformly distributed over \( 2N - 1 \) PEs
  - Probability that replica-group \( i \) is hit by failure: \( 1/(2N - 1) \)
  - Probability that replica-group \( \neq i \) is hit by failure: \( 2/(2N - 1) \)
  - Failure **not** uniformly distributed over replica-groups:
    - this is **not** the birthday problem
Differences with birthday problem

- $N$ processes; each replicated twice
- Uniform distribution of failures
- First failure: each replica-group has probability $1/N$ to be hit
- Second failure cannot hit failed PE
  - Failure uniformly distributed over $2N - 1$ PEs
  - Probability that replica-group $i$ is hit by failure: $1/(2N - 1)$
  - Probability that replica-group $\neq i$ is hit by failure: $2/(2N - 1)$
  - Failure not uniformly distributed over replica-groups: this is not the birthday problem
Differences with birthday problem

- \(N\) processes; each replicated twice
- Uniform distribution of failures
- First failure: each replica-group has probability \(1/N\) to be hit
- Second failure can hit failed PE
Differences with birthday problem

- $N$ processes; each replicated twice
- Uniform distribution of failures
- First failure: each replica-group has probability $1/N$ to be hit
- Second failure *can* hit failed PE
  - Suppose failure hits replica-group $i$
  - If failure hits failed PE: application survives
  - If failure hits running PE: application killed
- Not all failures hitting the same replica-group are equal: this is not the birthday problem
Differences with birthday problem

- $N$ processes; each replicated twice
- Uniform distribution of failures
- First failure: each replica-group has probability $1/N$ to be hit
- Second failure can hit failed PE
  - Suppose failure hits replica-group $i$
  - If failure hits failed PE: application survives
  - If failure hits running PE: application killed
- Not all failures hitting the same replica-group are equal: this is not the birthday problem
Differences with birthday problem

- $N$ processes; each replicated twice
- Uniform distribution of failures
- First failure: each replica-group has probability $1/N$ to be hit
- Second failure can hit failed PE
  - Suppose failure hits replica-group $i$
  - If failure hits failed PE: application survives
  - If failure hits running PE: application killed
- Not all failures hitting the same replica-group are equal: this is not the birthday problem
Correct analogy

\[ \hbox{\begin{array}{ccccccc}
\square & \square & \square & \square & \square & \ldots & \square \\
1 & 2 & 3 & 4 & \ldots & n \\
\bullet & \bullet & \bullet & \bullet & \bullet & \bullet & \bullet & \bullet & \bullet & \ldots
\end{array}} \]

\[ N \text{ bins, red and blue balls} \]

Mean Number of Failures to Interruption (bring down application)

\[ MNFTI = \text{expected number of balls to throw until one bin gets one ball of each color} \]
Theorem: $MNFTI = \mathbb{E}(NFTI|0)$ where

$$\mathbb{E}(NFTI|n_f) = \begin{cases} 
2 \cdot \frac{2N}{2N-n_f} + \frac{2N-2n_f}{2N-n_f} \mathbb{E}(NFTI|n_f + 1) 
& \text{if } n_f = N, \\
& \text{otherwise.}
\end{cases}$$

$\mathbb{E}(NFTI|n_f)$: expectation of number of failures to kill application, knowing that

- application is still running
- failures have already hit $n_f$ different replica-groups
Comparison

- 2N processors, no replication
  \[ \text{Throughput}_{\text{Std}} = 2N(1 - \text{Waste}) = 2N \left(1 - \sqrt{\frac{2C}{\mu_{2N}}} \right) \]

- N replica-pairs
  \[ \text{Throughput}_{\text{Rep}} = N \left(1 - \sqrt{\frac{2C}{\mu_{\text{rep}}}} \right) \]
  \[ \mu_{\text{rep}} = MNFTI \times \mu_{2N} = MNFTI \times \frac{\mu}{2N} \]

- Platform with 2N = 2^{20} processors \(\Rightarrow\) MNFTI = 1284.4
  \[ \mu = 10 \text{ years} \Rightarrow \text{better if } C \text{ shorter than 6 minutes} \]
Failure distribution

(a) Exponential

(b) Weibull, $k = 0.7$

Crossover point for replication when $\mu_{\text{ind}} = 125$ years
Weibull distribution with \( k = 0.7 \)

Dashed line: Ferreira et al.  
Solid line: Correct analogy

- Study by Ferreira et al. favors replication
- Replication beneficial if small \( \mu \) + large \( C \) + big \( p_{total} \)
Outline

1. Probabilistic models
2. In-memory checkpointing
3. Probabilistic models for advanced methods
4. Forward-recovery techniques
   - ABFT for Linear Algebra applications
   - Composite approach: ABFT & Checkpointing
5. Conclusion
Backward Recovery

- Rollback / Backward-recovery: returns in the history to recover from failures
- Spends time to re-execute computations
- Rebuilds states already reached
- Typical: checkpointing techniques
Forward-recovery

- Forward-recovery: proceeds without returning
- Pays additional costs during (failure-free) computation to maintain consistent redundancy
- Or pays additional computations when failures happen
- General technique: Replication
- Application-Specific techniques: Iterative algorithms with fixed point convergence, ABFT, ...
Outline

1. Probabilistic models
2. In-memory checkpointing
3. Probabilistic models for advanced methods
4. Forward-recovery techniques
   - ABFT for Linear Algebra applications
   - Composite approach: ABFT & Checkpointing
5. Conclusion
Example: block LU/QR factorization

- Solve $A \cdot x = b$ (hard)
- Transform $A$ into a $LU$ factorization
- Solve $L \cdot y = b$, then $U \cdot x = y$
Example: block LU/QR factorization

GETF2: factorize a column block
TRSM - Update row block
GEMM: Update the trailing matrix

- Solve $A \cdot x = b$ (hard)
- Transform $A$ into a $LU$ factorization
- Solve $L \cdot y = b$, then $U \cdot x = y$
Example: block LU/QR factorization

TRSM - Update row block

GETF2: factorize a column block

GEMM: Update the trailing matrix

- Solve $A \cdot x = b$ (hard)
- Transform $A$ into a $LU$ factorization
- Solve $L \cdot y = b$, then $U \cdot x = y$
Example: block LU/QR factorization

- 2D Block Cyclic Distribution (here $2 \times 3$)
- A single failure $\Rightarrow$ many data lost
Algorithm Based Fault Tolerant QR decomposition

- Checksum: invertible operation on the data of the row / column
- Checksum blocks are doubled, to allow recovery when data and checksum are lost together
Algorithm Based Fault Tolerant QR decomposition

- Checksum: invertible operation on the data of the row / column
- Checksum replication can be avoided by dedicating computing resources to checksum storage
Algorithm Based Fault Tolerant QR decomposition

- Checkpoint the next set of Q- Panels to be able to return to it in case of failures.
Algorithm Based Fault Tolerant QR decomposition

- Idea of ABFT: applying the operation on data and checksum preserves the checksum properties
For the part of the data that is not updated this way, the checksum must be re-calculated.
Algorithm Based Fault Tolerant QR decomposition

- To avoid slowing down all processors and panel operation, group checksum updates every $Q$ block columns.
Algorithm Based Fault Tolerant QR decomposition

To avoid slowing down all processors and panel operation, group checksum updates every $Q$ block columns.
Algorithm Based Fault Tolerant QR decomposition

To avoid slowing down all processors and panel operation, group checksum updates every $Q$ block columns.
Algorithm Based Fault Tolerant QR decomposition

- Then, update the missing coverage. Keep checkpoint block column to cover failures during that time.
In case of failure, conclude the operation, then
- Missing Data = Checksum - Sum(Existing Data)
Algorithm Based Fault Tolerant QR decomposition

- In case of failure, conclude the operation, then
  - Missing Checksum = Sum(Existing Data)
Algorithm Based Fault Tolerant QR decomposition

Failures may happen while inside a $Q$–panel factorization
Valid Checksum Information allows to recover most of the missing data, but not all: the checksum for the current $Q$—panels are not valid.
We use the checkpoint to restore the $Q$–panel in its initial state.
Algorithm Based Fault Tolerant QR decomposition

<table>
<thead>
<tr>
<th>M</th>
<th>P</th>
<th>mb</th>
<th>nb</th>
<th>Q</th>
<th>N</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>2</td>
<td>4</td>
<td>0</td>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>1</td>
<td>3</td>
<td>5</td>
<td>1</td>
<td>3</td>
<td>5</td>
</tr>
<tr>
<td>0</td>
<td>2</td>
<td>4</td>
<td>0</td>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>1</td>
<td>3</td>
<td>5</td>
<td>1</td>
<td>3</td>
<td>5</td>
</tr>
<tr>
<td>0</td>
<td>2</td>
<td>4</td>
<td>0</td>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>1</td>
<td>3</td>
<td>5</td>
<td>1</td>
<td>3</td>
<td>5</td>
</tr>
</tbody>
</table>

"Checkpoint"

<table>
<thead>
<tr>
<th>1</th>
<th>3</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>1</td>
<td>3</td>
<td>5</td>
</tr>
</tbody>
</table>

- and re-execute that part of the factorization, without applying outside of the scope
ABFT LU decomposition: implementation

MPI Implementation

- PBLAS-based: need to provide “Fault-Aware” version of the library
- Cannot enter recovery state at any point in time: need to complete ongoing operations despite failures
  - Recovery starts by defining the position of each process in the factorization and bring them all in a consistent state (checksum property holds)
- Need to test the return code of each and every MPI-related call
ABFT QR decomposition: performance

Figure 11. Weak scalability of FT-QR: run time overhead on Kraken when failures strike. Local snapshots have to be used along with re-factorization to recover the lost data and restore the matrix state. This is referred to as the “failure within Q panels.”

For example, when $Q = 12$, the failure is injected when the trailing update for the step with panel $(1301,1301)$ finishes. From the result in Figure 10, the recovery procedure in both cases adds a small overhead that also decreases when scaled to large problem size and process grid. For largest setups, only 2-3 percent of the execution time is spent recovering from a failure.

7.4. Extension to Other factorization

The algorithm proposed in this work can be applied to a wide range of dense matrix factorizations other than LU. As a demonstration, we have extended the fault tolerance functions to the ScaLAPACK QR factorization in double precision. Since QR uses a block algorithm similar to LU (and also similar to Cholesky), the integration of fault tolerance functions is mostly straightforward. Figure 11 shows the performance of QR with and without recovery. The overhead drops as the problem and grid size increase, although it remains higher than that of LU for the same problem size. This is expected: as the QR algorithm has a higher complexity than LU ($4N^3$ v.s. $2N^3$), the ABFT approach incurs more extra computation when updating checksums. Similar to the LU result, recovery adds an extra 2% overhead. At size 160,000, a failure incurs about 5.7% penalty to be recovered. This overhead becomes lower, the larger the problem or processor grid size considered.

Open MPI with ULFM; Kraken supercomputer;
As supercomputers grow ever larger in scale, the Mean Time to Failure becomes shorter and shorter, making the complete and successful execution of complex applications more and more difficult.

FT-LA delivers a new approach, utilizing Algorithm-Based Fault Tolerance (ABFT), to help factorization algorithms survive fail-stop failures.

The FT-LA software package extends ScaLAPACK with ABFT routines, and in sharp contrast with legacy checkpoint-based approaches, ABFT does not incur I/O overhead, and promises a much more scalable protection scheme.

Cost of ABFT comes only from extra flops (to update checksums) and extra storage. Cost decreases with machine scale (divided by \(\frac{P}{Q}\) when using \(P \times Q\) processes).

Matrix protected by block row checksum. The algorithm updates both the trailing matrix AND the checksums.

Missing blocks reconstructed by inverting the checksum operation.

**FUNCTIONALITY**
- Linear Systems of Equations
- Least Squares
- Cholesky, LU
- QR (with protection of the upper and lower factors)

**FEATURES**
- WORK IN PROGRESS
  - Covering four precisions: double complex, single complex, double real, single real (ZCDS)
  - Deploys on MPI FT draft (ULFM), or with "Checkpoint-on-failure"
  - Allows toleration of permanent crashes

**PROTECTION**
- Hessenber Reduction, Soft (silent) Errors

**FIND OUT MORE AT**
[http://icl.cs.utk.edu/ft-la](http://icl.cs.utk.edu/ft-la)

As supercomputers grow ever larger in scale, the Mean Time to Failure becomes shorter and shorter, making the complete and successful execution of complex applications more and more difficult.

**MPI-Next ULFM Performance**

- **Open MPI with ULFM; Kraken supercomputer;**
ABFT QR decomposition: performance

Fig. 2. ABFT QR and one CoF recovery on Kraken (Lustre).

Fig. 3. ABFT QR and one CoF recovery on Dancer (local SSD).

Fig. 4. Time breakdown of one CoF recovery on Dancer (local SSD).

Checkpoint on Failure - MPI Performance

Open MPI; Kraken supercomputer;
Outline

1. Probabilistic models
2. In-memory checkpointing
3. Probabilistic models for advanced methods
4. Forward-recovery techniques
   - ABFT for Linear Algebra applications
   - Composite approach: ABFT & Checkpointing
5. Conclusion
Fault tolerance techniques

General techniques

- Replication
- Rollback recovery
  - Coordinated checkpointing
  - Uncoordinated checkpointing & Message logging
  - Hierarchical checkpointing

Application-specific techniques

- Algorithm Based Fault Tolerance (ABFT)
- Iterative convergence
- Approximated computation
**Typical Application**

```c
for ( aninsanenumber ) {
    /* Extract data from simulation, fill up matrix */
    sim2mat();

    /* Factorize matrix, Solve */
    dgeqrf();
    dsolve();

    /* Update simulation with result vector */
    vec2sim();
}
```

**Characteristics**

- Large part of (total) computation spent in factorization/solve
- Between LA operations:
  - Use resulting vector / matrix with operations that do not preserve the checksums on the data
  - Modify data not covered by ABFT algorithms
Typical Application

```c
for ( aninsanenumber ) {
    /* Extract data */
    * simulation ,
    * matrix */
    sim2mat();

    /* Factorize matrix */
    * Solve */
    dgeqrf();
    dsolve();

    /* Update simulation */
    * with result vector */
    vec2sim();
}
```

**Goodbye ABFT?!**

- Large part of (total) computation spent in factorization/solve
- Between LA operations:
  - use resulting vector / matrix with operations that do not preserve the checksums on the data
  - modify data not covered by ABFT algorithms
Application

Problem Statement

Typical Application

```c
for ( a = 0; a < aninsanenumbe r; a++) {
    /* Extract data from simulation, fill up matrix */
    sim2mat();
    /* Factorize matrix, Solve */
   dgeqrf();
    dsolve();
    /* Update simulation with result vector */
    vec2sim();
}
```

How to use fault tolerant operations(*) within a non-fault tolerant(**) application?(***)

(*) ABFT, or other application-specific FT
(**) Or within an application that does not have the same kind of FT
(***) And keep the application globally fault tolerant...

😊 use resulting vector / matrix with operations that do not preserve the checksums on the data

😢 modify data not covered by ABFT algorithms
ABFT&PeriodicCkpt: no failure

Periodic Checkpoint

Process 0

Process 1

Process 2

Application

Library

Split Forced Checkpoints
ABFT & Periodic Checkpoint: failure during Library phase

Process 0
Process 1
Process 2

Failure (during Library)
Rollback (partial)
Recovery
ABFT Recovery

Application
Library
Application
Library
Application
Library

Anne.Benoit@ens-lyon.fr
ABFT&PeriodicCkpt: failure during General phase

Process 0

Process 1

Process 2

Failure (during General)

Rollback (full)

Recovery

Application

Library

Application

Library

Application

Library

Anne.Benoit@ens-lyon.fr

CR02

Fault tolerance (3)
ABFT\&PeriodicCkpt: Optimizations

- If the duration of the \texttt{GENERAL} phase is too small: don’t add checkpoints
- If the duration of the \texttt{LIBRARY} phase is too small: don’t do ABFT recovery, remain in \texttt{GENERAL} mode
  - this assumes a performance model for the library call
ABFT\&PERIODICCKPT: Optimizations

- If the duration of the **GENERAL** phase is too small: don’t add checkpoints
- If the duration of the **LIBRARY** phase is too small: don’t do ABFT recovery, remain in **GENERAL** mode
  - this assumes a performance model for the library call
A few notations

**Times, Periods**

- **$T_0$:** Duration of an Epoch (without FT)
- **$T_L = \alpha T_0$:** Time spent in the **Library** phase
- **$T_G = (1 - \alpha) T_0$:** Time spent in the **General** phase
- **$P_G$:** Periodic Checkpointing Period
- **$T_{ff}, T'_{ff}, T''_{ff}$:** “Fault Free” times
- **$t_{lost_G}, t_{lost_L}$:** Lost time (recovery overloads)
- **$T_{final_G}, T_{final_L}$:** Total times (with faults)
A few notations

Costs

\( C_L = \rho C \): time to take a checkpoint of the \textit{Library} data set
\( C_{\bar{L}} = (1 - \rho)C \): time to take a checkpoint of the \textit{General} data set
\( R, R_{\bar{L}} \): time to load a full / \textit{General} data set checkpoint
\( D \): down time (time to allocate a new machine / reboot)
\( \text{Recons}_{\text{ABFT}} \): time to apply the ABFT recovery
\( \phi \): Slowdown factor on the \textit{Library} phase, when applying ABFT
Overall

Time (with overheads) of \textbf{Library} phase is constant (in $P_G$):

$$T_L^{\text{final}} = \frac{1}{1 - \frac{D + R_L + \text{Recons}_{ABFT}}{\mu}} \times (\alpha \times T_L + C_L)$$

Time (with overheads) of \textbf{General} phase accepts two cases:

$$T_G^{\text{final}} = \begin{cases} 
\frac{1}{1 - \frac{D + R + \frac{C_L}{2}}{\mu}} \times (T_G + C_L) & \text{if } T_G < P_G \\
\frac{1}{(1 - \frac{C}{P_G})(1 - \frac{D + R + \frac{P_G}{2}}{\mu})} & \text{if } T_G \geq P_G 
\end{cases}$$

Which is minimal in the second case, if

$$P_G = \sqrt{2C(\mu - D - R)}$$
From the previous, we derive the waste, which is obtained by

\[ \text{WASTE} = 1 - \frac{T_0}{T_{\text{final}}^G + T_{\text{final}}^L} \]
Toward Exascale, and beyond!

Let's think at scale

- Number of components $\uparrow \Rightarrow$ MTBF $\downarrow$
- Number of components $\uparrow \Rightarrow$ Problem size $\uparrow$
- Problem size $\uparrow \Rightarrow$
  
  Computation time spent in Library phase $\uparrow$

😀 ABFT&PERIODICCKPT should perform better with scale
❔ By how much?
Competitors

FT algorithms compared

**PeriodicCkpt**  Basic periodic checkpointing

**Bi-PeriodicCkpt**  Applies incremental checkpointing techniques to save only the library data during the library phase

**ABFT&PeriodicCkpt**  The algorithm described above
Weak Scale Scenario #1

- Number of components, $n$, increase
- Memory per component remains constant
- Problem size increases in $O(\sqrt{n})$ (e.g. matrix operation)

- $\mu$ at $n = 10^5$: 1 day, is in $O\left(\frac{1}{n}\right)$
- $C (=R)$ at $n = 10^5$, is 1 minute, is in $O(n)$
- $\alpha$ is constant at 0.8, as is $\rho$.
  (both Library and General phase increase in time at the same speed)
Weak Scale #1

![Graph showing fault tolerance and waste versus nodes for different checkpointing strategies: PeriodicCheck, Bi-PeriodicCheck, and ABFT Checkpoint.](image)

### Table

<table>
<thead>
<tr>
<th># Faults</th>
<th>PeriodicCheck</th>
<th>Bi-PeriodicCheck</th>
<th>ABFT Checkpoint</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0.05</td>
<td>0.1</td>
<td>0.15</td>
</tr>
<tr>
<td>100k</td>
<td>0.2</td>
<td>0.25</td>
<td>0.3</td>
</tr>
<tr>
<td>1M</td>
<td>0.3</td>
<td>0.35</td>
<td>0.4</td>
</tr>
</tbody>
</table>

### Fault Tolerance (3)

Anne.Benoit@ens-lyon.fr  
CR02  
Fault tolerance (3)
Weak Scale Scenario #2

- Number of components, \( n \), increase
- Memory per component remains constant
- Problem size increases in \( O(\sqrt{n}) \) (e.g. matrix operation)

- \( \mu \) at \( n = 10^5 \): 1 day, is \( O(\frac{1}{n}) \)
- \( C \) (\( = R \)) at \( n = 10^5 \), is 1 minute, is in \( O(n) \)
- \( \rho \) remains constant at 0.8, but Library phase is \( O(n^3) \) when General phases progresses in \( O(n^2) \) (\( \alpha \) is 0.8 at \( n = 10^5 \) nodes).
Weak Scale #2

Anne.Benoit@ens-lyon.fr  CR02  Fault tolerance (3)
Weak Scale Scenario #3

- Number of components, \( n \), increase
- Memory per component remains constant
- Problem size increases in \( O(\sqrt{n}) \) (e.g. matrix operation)

- \( \mu \) at \( n = 10^5 \): 1 day, is \( O(\frac{1}{n}) \)
- \( C (=R) \) at \( n = 10^5 \), is 1 minute, stays independent of \( n \) (\( O(1) \))
- \( \rho \) remains constant at 0.8, but \text{Library} phase is \( O(n^3) \) when \text{General} phases progresses in \( O(n^2) \) (\( \alpha \) is 0.8 at \( n = 10^5 \) nodes).
Weak Scale #3

- Nb Faults PeriodicCkpt
- Nb Faults Bi-PeriodicCkpt
- Nb Faults ABFT PeriodicCkpt

Waste Nodes:
- PeriodicCkpt
- Bi-PeriodicCkpt
- ABFT PeriodicCkpt

Fault tolerance (3)
Outline

1. Probabilistic models
2. In-memory checkpointing
3. Probabilistic models for advanced methods
4. Forward-recovery techniques
5. Conclusion
Leitmotiv

Resilient research on resilience

Models needed to assess techniques at scale without bias 😊
Conclusion

- Multiple approaches to Fault Tolerance
- Application-Specific Fault Tolerance will always provide more benefits:
  - Checkpoint Size Reduction (when needed)
  - Portability (can run on different hardware, different deployment, etc.)
  - Diversity of use (can be used to restart the execution and change parameters in the middle)
Conclusion

- Multiple approaches to Fault Tolerance
- General Purpose Fault Tolerance is a required feature of the platforms
  - Not every computer scientist needs to learn how to write fault-tolerant applications
  - Not all parallel applications can be ported to a fault-tolerant version
- Faults are a feature of the platform. Why should it be the role of the programmers to handle them?
Conclusion

Application-Specific Fault Tolerance

- Fault Tolerance is introducing redundancy in the application
  - replication of computation
  - maintaining invariant in the data
- Requirements of a more Fault-friendly programming environment
  - MPI-Next evolution
  - Other programming environments?
Conclusion

General Purpose Fault Tolerance

- Software/hardware techniques to reduce checkpoint, recovery, migration times and to improve failure prediction
- Multi-criteria scheduling problem
  execution time/energy/reliability
  add replication
  best resource usage (performance trade-offs)
- Need combine all these approaches!

Several challenging algorithmic/scheduling problems 😊
Bibliography

Exascale
- Toward Exascale Resilience, Cappello F. et al., IJHPCA 23, 4 (2009)
- The International Exascale Software Roadmap, Dongarra, J., Beckman, P. et al., IJHPCA 25, 1 (2011)

ABFT Algorithm-based fault tolerance applied to high performance computing, Bosilca G. et al., JPDC 69, 4 (2009)


Replication Evaluating the viability of process replication reliability for exascale systems, Ferreira K. et al, SC’2011

Models
- Checkpointing strategies for parallel jobs, Bougeret M. et al., SC’2011
- Unified model for assessing checkpointing protocols at extreme-scale, Bosilca G et al., INRIA RR-7950, 2012
Bibliography

New Monograph, Springer Verlag 2015

Thanks Yves Robert, Thomas Hérault, George Bosilca and Aurélien Bouteiller for the slides (SC’15 tutorial)