Unified Model for Assessing Checkpointing Protocols at Extreme-Scale

George Bosilca\textsuperscript{1}, Aurélien Bouteiller\textsuperscript{1}, Elisabeth Brunet\textsuperscript{2}, Franck Cappello\textsuperscript{3}, Jack Dongarra\textsuperscript{1}, Amina Guermouche\textsuperscript{4}, Thomas Hérault\textsuperscript{1}, Yves Robert\textsuperscript{1,4}, Frédéric Vivien\textsuperscript{4}, and Dounia Zaidouni\textsuperscript{4}

1. University of Tennessee Knoxville, USA
2. Telecom SudParis, France
3. INRIA & University of Illinois at Urbana Champaign, USA
4. Ecole Normale Supérieure de Lyon & INRIA, France

Pittsburgh, June 28, 2012
Motivation

• **Very very** large number of processing elements (e.g., $2^{20}$)  
  $\implies$ Probability of failures dramatically increases

• Large application to be executed on whole platform  
  $\implies$ Failure(s) will most likely occur before completion!

• Resilience provided through checkpointing
  1. Coordinated protocols
  2. Hierarchical protocols
Which checkpointing protocol to use?

Coordinated checkpointing

😊 No risk of cascading rollbacks
😊 No need to log messages
😢 All processors need to roll back
😢 Rumor: May not scale to very large platforms

Hierarchical checkpointing

😢 Need to log inter-groups messages
  ● Slowdowns failure-free execution
  ● Increases checkpoint size/time
😊 Only processors from failed group need to roll back
😊 Faster re-execution with logged messages
😊 Rumor: Should scale to very large platforms
Outline

1. Protocol Overhead
   - Coordinated checkpointing
   - Hierarchical checkpointing

2. Accounting for message logging

3. Instanciating the model
   - Applications
   - Platforms

4. Experimental results
   - Plotting formulas
   - Simulations
Outline

1. Protocol Overhead
2. Accounting for message logging
3. Instanciating the model
4. Experimental results
Framework

- Periodic checkpointing policies (of period $T$)
- Independent and identically distributed failures
- Platform failure inter-arrival time: $\mu$
- Tightly-coupled application:
  \[ \text{progress } \Leftrightarrow \text{all processors available} \]
- First-order approximation: at most one failure within a period

**Waste**: fraction of time not spent for useful computations
Checkpointing cost

- Time spent working
- Time spent checkpointing

- Computing the first chunk
- Checkpointing the first chunk
- Processing the first chunk
- Processing the second chunk
Checkpointing cost

**Blocking model:** while a checkpoint is taken, no computation can be performed
Non-blocking model: while a checkpoint is taken, computations are not impacted (e.g., first copy state to RAM, then copy RAM to disk)
**General model:** while a checkpoint is taken, computations are slowed-down: during a checkpoint of duration $C$, the same amount of computation is done as during a time $\alpha C$ without checkpointing ($0 \leq \alpha \leq 1$).
1 Protocol Overhead
   Coordinated checkpointing
   Hierarchical checkpointing

2 Accounting for message logging

3 Instanciating the model
   Applications
   Platforms

4 Experimental results
   Plotting formulas
   Simulations
Waste in absence of failures

\[
\begin{align*}
\text{Time elapsed since last checkpoint: } & \quad T \\
\text{Amount of computation saved: } & \quad (T - C) + \alpha C \\
WASTE_{\text{coord-nofailure}} = & \quad \frac{T - ((T - C) + \alpha C)}{T} = \frac{(1 - \alpha)C}{T}
\end{align*}
\]
Waste due to failures

Failure can happen

1. During computation phase
2. During checkpointing phase
Waste due to failures

- Time spent working
- Time spent checkpointing
- Time spent working with slowdown

$P_0$, $P_1$, $P_2$, $P_3$
Waste due to failures

Time spent working  Time spent checkpointing  Time spent working with slowdown

$P_0$  $P_1$  $P_2$  $P_3$
Waste due to failures

Coordinated checkpointing protocol: when one processor is victim of a failure, all processors lose their work and must roll back to last checkpoint
Waste due to failures in computation phase

- Time spent working
- Time spent checkpointing
- Time spent working with slowdown
- Downtime

$P_0$, $P_1$, $P_2$, $P_3$
Waste due to failures in computation phase

Coordinated checkpointing protocol: All processors must recover from last checkpoint
Waste due to failures in computation phase

Redo the work destroyed by the failure, that was done in the checkpointing phase before the computation phase

But no checkpoint is taken in parallel, hence this re-computation is faster than the original computation
Waste due to failures in computation phase

Re-execute the computation phase
Waste due to failures in computation phase

Finally, the checkpointing phase is executed

First-order approximation: we assume that no other failure occurs during the re-execution
Waste due to failures in computation phase

\[ \Delta - T = T_{lost} + \alpha C \]

**Expectation:** \[ T_{lost} = \frac{1}{2}(T - C) \]

\[ \text{RE-EXEC}_{\text{coord-fail-in-work}} = \frac{T - C}{2} + \alpha C \]
Waste due to failures

• Failure in the computation phase (probability: \( \frac{T - C}{T} \))

\[
\text{RE-EXEC}_{\text{coord-fail-in-work}} = \frac{T - C}{2} + \alpha C
\]

• Failure in the checkpointing phase (probability: \( \frac{C}{T} \))

\[
\text{RE-EXEC}_{\text{coord-fail-in-checkpoint}} = T - \frac{C}{2} + \alpha C
\]

\[
\frac{T - C}{T} \left( \frac{T - C}{2} + \alpha C \right) + \frac{C}{T} \left( T - \frac{C}{2} + \alpha C \right) = \alpha C + \frac{T}{2}
\]
Overall waste

\[ \text{WASTE}_{\text{coord}} = \text{WASTE}_{\text{coord-nofailure}} + \frac{1}{\mu} (D + R + \text{RE-EXEC}_{\text{coord}}) \]

\[ = \frac{(1 - \alpha)C}{T} + \frac{1}{\mu} \left( D + R + \alpha C + \frac{T}{2} \right) \]

Minimize \( \text{WASTE}_{\text{coord}} \) subject to:

- \( C \leq T \) (by construction)
- \( T \leq 0.1\mu \) \( \Rightarrow \text{Proba}(\text{Poisson}(\frac{T}{\mu}) \geq 2) \leq 0.005 \)
1 Protocol Overhead
   Coordinated checkpointing
   Hierarchical checkpointing

2 Accounting for message logging

3 Instanciating the model
   Applications
   Platforms

4 Experimental results
   Plotting formulas
   Simulations
Hierarchical checkpointing

- Processors partitioned into $G$ groups
- Each group includes $q$ processors
- Inside each group: coordinated checkpointing in time $C(q)$
- Inter-group messages are logged
Impact of checkpointing

When a group checkpoints, its own computation speed is slowed-down

This holds for all groups because of the tightly-coupled assumption

\[
\text{WASTE} = \frac{T - \text{WORK}}{T} \quad \text{where} \quad \text{WORK} = T - (1 - \alpha)GC(q)
\]
Impact of checkpointing

- Time spent working
- Time spent checkpointing
- Time spent working with slowdown
- Downtime
- Recovery time
- Re-executing slowed-down work

G1
G2
G3
G4
G5
Impact of checkpointing

- Time spent working
- Time spent checkpointing
- Time spent working with slowdown
- Downtime
- Recovery time
- Re-executing slowed-down work

**Diagram:**

- $G_1$
- $G_2$
- $G_3$
- $G_4$
- $G_5$
Impact of checkpointing

Tightly-coupled model: while one group is in downtime, none can work
Failure during computation phase

Tightly-coupled model: while one group is in recovery, none can work
Failure during computation phase

Groups must have completed the same amount of work in between two consecutive checkpoints, independently of the fact that a failure may or may not have happened on the platform in between these checkpoints. Hence, no checkpointing is possible during the rollback.
Failure during computation phase

Redo work done during previous checkpointing phase and that was destroyed by the failure
Failure during computation phase

Redo work done during previous checkpointing phase and that was destroyed by the failure

But no checkpoint is taken in parallel, hence this re-computation is faster than the original computation
Failure during computation phase

Redo work done in computation phase and that was destroyed by the failure
Failing group has reached the point where it previously failed, all groups now resume execution in parallel and complete the computation phase.
Failure during computation phase

Finally, perform checkpointing phase
Failure during computation phase

\[ T_{\text{lost}} + \alpha(G - g + 1)C \]

Expectation: \[ T_{\text{lost}} = \frac{1}{2}(T - G.C) \]

Approximated RE-EXEC: \[ \frac{T - G.C}{2} + \alpha(G - g + 1)C \]
Failure during computation phase

Approximated RE-EXEC: \( \frac{T - G.C}{2} + \alpha(G - g + 1)C \)

Average approximated RE-EXEC:

\[
\frac{1}{G} \sum_{g=1}^{G} \left[ \frac{T - G.C(q)}{2} + \alpha(G - g + 1)C(q) \right] = \frac{T - G.C(q)}{2} + \alpha \frac{G + 1}{2} C
\]
Failure during checkpointing phase

- Time spent working
- Time spent checkpointing
- Time spent working with slowdown
- Downtime
- Recovery time
- Re-executing slowed-down work

Diagram showing the timeline for different groups ($G_1$ to $G_5$) with labeled phases.
Failure during checkpointing phase

When does the failing group fail?

1. Before starting its own checkpoint
2. While taking its own checkpoint
3. After completing its own checkpoint
Average waste for failures during checkpointing phase

Average $\text{RE-EXEC}$ when the failing-group $g$ fails

Overall average $\text{RE-EXEC}$: $\text{RE-EXEC}_{\text{ckpt}} =$

$$
\frac{1}{G} ((g - 1).\text{RE-EXEC}_{\text{before ckpt}} + 1.\text{RE-EXEC}_{\text{during ckpt}} + (G - g).\text{RE-EXEC}_{\text{after ckpt}})
$$

Average over all groups:

$$
\text{AVG}_G \text{RE-EXEC}_{\text{ckpt}} =
\frac{G + 1}{2G} T + \frac{\alpha C(q)(G + 3)}{2} + \frac{C(q)(1 - 2\alpha)}{2G} - \frac{C(q)(G + 1)}{2}
$$
Average waste

\[ WASTE_{hierarch} = \frac{T - WORK}{T} + \frac{1}{\mu} \left( D(q) + R(q) + RE-EXEC \right) \]

\[ = \frac{1}{2\mu T} \times \left( \begin{array}{c} T^2 \\ + GC(q) [(1 - \alpha)(2\mu - T) + (2\alpha - 1)C(q)] \\ + T [2(D(q) + R(q)) + (\alpha + 1)C(q)] \\ + (1 - 2\alpha)C(q)^2 \end{array} \right) \]

Minimize \( WASTE_{hierarch} \) subject to:

- \( GC(q) \leq T \) (by construction)
- \( T \leq 0.1\mu \) (\( \Rightarrow \) \( \text{Proba}(\text{Poisson}(\frac{T}{\mu}) \geq 2) \leq 0.005 \))
Outline

1. Protocol Overhead
2. Accounting for message logging
3. Instanciating the model
4. Experimental results
Impact on work

- 😞 Logging messages slows down execution:
  \[ \text{WORK becomes } \lambda \text{WORK}, \text{ where } 0 < \lambda < 1 \]
  Typical value: \( \lambda \approx 0.98 \)

- ☺ Re-execution after a failure is faster:
  \[ \text{RE-EXEC becomes } \frac{\text{RE-EXEC}}{\rho}, \text{ where } \rho \in [1..2] \]
  Typical value: \( \rho \approx 1.5 \)

\[
\text{WASTE}_{\text{hierarch}} = \frac{T - \lambda \text{WORK}}{T} + \frac{1}{\mu} \left( D(q) + R(q) + \frac{\text{RE-EXEC}}{\rho} \right)
\]
Impact on checkpoint size

- Inter-groups messages logged continuously
- Checkpoint size increases with amount of work executed before a checkpoint
- $C_0(q)$: Checkpoint size of a group without message logging

\[
C(q) = C_0(q)(1 + \beta \text{WORK}) \iff \beta = \frac{C(q) - C_0(q)}{C_0(q) \text{WORK}}
\]

\[
\text{WORK} = \lambda(T - (1 - \alpha)GC(q))
\]

\[
C(q) = \frac{C_0(q)(1 + \beta \lambda T)}{1 + GC_0(q) \beta \lambda (1 - \alpha)}
\]

- Constraint $GC(q) \leq T$ translates into

\[
GC_0(q) \beta \lambda \alpha \leq 1 \text{ and } T \geq \frac{GC_0(q)}{1 - GC_0(q) \beta \lambda \alpha}
\]
Outline

1. Protocol Overhead
2. Accounting for message logging
3. Instanciating the model
4. Experimental results
Three case studies

**Coord-IO**

Coordinated approach: \( C = C_{\text{Mem}} = \frac{\text{Mem}}{b_{\text{io}}} \)

where Mem is the memory footprint of the application

**Hierarch-IO**

Several (large) groups, *I/O-saturated*

⇒ groups checkpoint sequentially

\[
C_0(q) = \frac{C_{\text{Mem}}}{G} = \frac{\text{Mem}}{G b_{\text{io}}}
\]

**Hierarch-Port**

Very large number of smaller groups, *port-saturated*

⇒ some groups checkpoint in parallel

Groups of \( q_{\text{min}} \) processors, where \( q_{\text{min}} b_{\text{port}} \geq b_{\text{io}} \)
1 Protocol Overhead
   Coordinated checkpointing
   Hierarchical checkpointing

2 Accounting for message logging

3 Instanciating the model
   Applications
   Platforms

4 Experimental results
   Plotting formulas
   Simulations
Three applications

1. 2D-stencil
2. 3D-Stencil
   - Plane
   - Line
3. Matrix product
Computing $\beta$ for Stencil-2D

\[ C(q) = C_0(q) + \text{Logged}_\text{Msg} = C_0(q)(1 + \beta \text{Work}) \]

Real $n \times n$ matrix and $p \times p$ grid

\[ \text{Work} = \frac{9b^2}{s_p}, \; b = n/p \]

Each process sends a block to its 4 neighbors
Computing $\beta$ for Stencil-2D

\[
C(q) = C_0(q) + \text{Logged}_\text{Msg} = C_0(q)(1 + \beta \text{Work})
\]

Real $n \times n$ matrix and $p \times p$ grid

\[
\text{Work} = \frac{9b^2}{s_p}, \quad b = \frac{n}{p}
\]

Each process sends a block to its 4 neighbors

**Hierarch-IO:**

- 1 group = 1 grid row
- 2 out of the 4 messages are logged
- $\beta = \frac{2s_p}{9b^3}$
Computing $\beta$ for Stencil-2D

$$C(q) = C_0(q) + \text{Logged}_\text{Msg} = C_0(q)(1 + \beta \text{Work})$$

Real $n \times n$ matrix and $p \times p$ grid

$$\text{Work} = \frac{9b^2}{sp}, \quad b = n/p$$

Each process sends a block to its 4 neighbors

**Hierarch-IO:**

- 1 group = 1 grid row
- 2 out of the 4 messages are logged
- $\beta = \frac{2sp}{9b^3}$

**Hierarch-Port:**

- $\beta$ doubles
Three applications: 2) 3D-stencil

- Real matrix of size $n \times n \times n$ partitioned across a $p \times p \times p$ processor grid
- Each processor holds a cube of size $b = n/p$
- At each iteration:
  - average each matrix element with its 27 closest neighbors
  - exchange the six faces of its cube

(Parallel) work for one iteration is $\text{WORK} = \frac{27b^3}{s_p}$

Three hierarchical variants

1. **Hierarch-IO-Plane**: group = horizontal plane of size $p^2$:
   \[
   \beta = \frac{2s_p}{27b^3}
   \]

2. **Hierarch-IO-Line**: group = horizontal line of size $p$:
   \[
   \beta = \frac{4s_p}{27b^3}
   \]

3. **Hierarch-Port**: groups of size $q_{min}$:
   \[
   \beta = \frac{6s_p}{27b^3}
   \]
1. Protocol Overhead
   Coordinated checkpointing
   Hierarchical checkpointing

2. Accounting for message logging

3. Instanciating the model
   Applications
   Platforms

4. Experimental results
   Plotting formulas
   Simulations
## Four platforms: basic characteristics

<table>
<thead>
<tr>
<th>Name</th>
<th>Number of cores</th>
<th>Number of processors $p_{total}$</th>
<th>Number of cores per processor</th>
<th>Memory per processor</th>
<th>I/O Network Bandwidth (b_{io})</th>
</tr>
</thead>
<tbody>
<tr>
<td>Titan</td>
<td>299,008</td>
<td>16,688</td>
<td>16</td>
<td>32GB</td>
<td>300GB/s 300GB/s</td>
</tr>
<tr>
<td>K-Computer</td>
<td>705,024</td>
<td>88,128</td>
<td>8</td>
<td>16GB</td>
<td>150GB/s 96GB/s</td>
</tr>
<tr>
<td>Exascale-Slim</td>
<td>1,000,000,000</td>
<td>1,000,000</td>
<td>1,000</td>
<td>64GB</td>
<td>1TB/s 1TB/s</td>
</tr>
<tr>
<td>Exascale-Fat</td>
<td>1,000,000,000</td>
<td>100,000</td>
<td>10,000</td>
<td>640GB</td>
<td>1TB/s 1TB/s</td>
</tr>
</tbody>
</table>
## Four platforms: 2D-Stencil and Matrix-Product

<table>
<thead>
<tr>
<th>Name</th>
<th>Scenario</th>
<th>( G(C(q)) )</th>
<th>( \beta ) for 2D-Stencil</th>
<th>( \beta ) for Matrix-Product</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Titan</strong></td>
<td>Coord-IO</td>
<td>1 (2,048s)</td>
<td>/</td>
<td>0.0004280</td>
</tr>
<tr>
<td></td>
<td>Hierarch-IO</td>
<td>136 (15s)</td>
<td>0.0001098</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Hierarch-Port</td>
<td>1,246 (1.6s)</td>
<td>0.0002196</td>
<td>0.0008561</td>
</tr>
<tr>
<td><strong>K-Computer</strong></td>
<td>Coord-IO</td>
<td>1 (14,688s)</td>
<td>/</td>
<td>0.001113</td>
</tr>
<tr>
<td></td>
<td>Hierarch-IO</td>
<td>296 (50s)</td>
<td>0.0002858</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Hierarch-Port</td>
<td>17,626 (0.83s)</td>
<td>0.0005716</td>
<td>0.002227</td>
</tr>
<tr>
<td><strong>Exascale-Slim</strong></td>
<td>Coord-IO</td>
<td>1 (64,000s)</td>
<td>/</td>
<td>0.001013</td>
</tr>
<tr>
<td></td>
<td>Hierarch-IO</td>
<td>1,000 (64s)</td>
<td>0.0002599</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Hierarch-Port</td>
<td>200,0000 (0.32s)</td>
<td>0.0005199</td>
<td>0.002026</td>
</tr>
<tr>
<td><strong>Exascale-Fat</strong></td>
<td>Coord-IO</td>
<td>1 (64,000s)</td>
<td>/</td>
<td>0.0003203</td>
</tr>
<tr>
<td></td>
<td>Hierarch-IO</td>
<td>316 (217s)</td>
<td>0.00008220</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Hierarch-Port</td>
<td>33,333 (1.92s)</td>
<td>0.00016440</td>
<td>0.0006407</td>
</tr>
</tbody>
</table>
## Four platforms: 3D-STENCIL

<table>
<thead>
<tr>
<th>Name</th>
<th>Scenario</th>
<th>$G$</th>
<th>$\beta$ for 3D-STENCIL</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Titan</strong></td>
<td>COORD-IO</td>
<td>1</td>
<td>/</td>
</tr>
<tr>
<td></td>
<td>HIERARCH-IO-PLANE</td>
<td>26</td>
<td>0.001476</td>
</tr>
<tr>
<td></td>
<td>HIERARCH-IO-LINE</td>
<td>675</td>
<td>0.002952</td>
</tr>
<tr>
<td></td>
<td>HIERARCH-PORT</td>
<td>1,246</td>
<td>0.004428</td>
</tr>
<tr>
<td><strong>K-Computer</strong></td>
<td>COORD-IO</td>
<td>1</td>
<td>/</td>
</tr>
<tr>
<td></td>
<td>HIERARCH-IO-PLANE</td>
<td>44</td>
<td>0.003422</td>
</tr>
<tr>
<td></td>
<td>HIERARCH-IO-LINE</td>
<td>1,936</td>
<td>0.006844</td>
</tr>
<tr>
<td></td>
<td>HIERARCH-PORT</td>
<td>17,626</td>
<td>0.010266</td>
</tr>
<tr>
<td><strong>Exascale-Slim</strong></td>
<td>COORD-IO</td>
<td>1</td>
<td>/</td>
</tr>
<tr>
<td></td>
<td>HIERARCH-IO-PLANE</td>
<td>100</td>
<td>0.003952</td>
</tr>
<tr>
<td></td>
<td>HIERARCH-IO-LINE</td>
<td>10,000</td>
<td>0.007904</td>
</tr>
<tr>
<td></td>
<td>HIERARCH-PORT</td>
<td>200,000</td>
<td>0.011856</td>
</tr>
<tr>
<td><strong>Exascale-Fat</strong></td>
<td>COORD-IO</td>
<td>1</td>
<td>/</td>
</tr>
<tr>
<td></td>
<td>HIERARCH-IO-PLANE</td>
<td>46</td>
<td>0.001834</td>
</tr>
<tr>
<td></td>
<td>HIERARCH-IO-LINE</td>
<td>2,116</td>
<td>0.003668</td>
</tr>
<tr>
<td></td>
<td>HIERARCH-PORT</td>
<td>33,333</td>
<td>0.005502</td>
</tr>
</tbody>
</table>
Outline

1. Protocol Overhead
2. Accounting for message logging
3. Instanciating the model
4. Experimental results
1 Protocol Overhead
   Coordinated checkpointing
   Hierarchical checkpointing

2 Accounting for message logging

3 Instanciating the model
   Applications
   Platforms

4 Experimental results
   Plotting formulas
   Simulations
Platform: Titan

Waste as a function of processor MTBF $\mu$
Platform: K-Computer

Waste as a function of processor MTBF $\mu$
Platform: Exascale

\textbf{WASTE} = 1 for all scenarios!!!
Platform: Exascale

\[ \text{WASTE} = 1 \text{ for all scenarios!!!} \]

Goodbye Exascale?!
Platform: Exascale with $C = 1,000$

Waste as a function of processor MTBF $\mu$, $C = 1,000$
Platform: Exascale with \( C = 100 \)

Waste as a function of processor MTBF \( \mu \), \( C = 100 \)
1. Protocol Overhead
   - Coordinated checkpointing
   - Hierarchical checkpointing

2. Accounting for message logging

3. Instanciating the model
   - Applications
   - Platforms

4. Experimental results
   - Plotting formulas
   - Simulations
Makespan (in days) as a function of processor MTBF $\mu$
Platform: Exascale with $C = 1,000$

Makespan (in days) as a function of processor MTBF $\mu$, $C = 1,000$
Platform: Exascale with $C = 100$

Makespan (in days) as a function of processor MTBF $\mu$, $C = 100$
Conclusion

- First attempt at analytical comparison of coordinated and hierarchical checkpointing protocols
- Classical models (Young, Daly) extended
  - Several new parameters ($\alpha$, $\lambda$, $\rho$)
  - Message logging impact ($\beta$)
- Instantiation
  - Scenarios: COORD-IO, HIERARCH-IO, HIERARCH-Port
  - Realistic application/platform combinations
- Future work: (partial) replication, prediction, energy