Fault tolerance techniques for high-performance computing

Part 2

Anne Benoit

ENS Lyon

Anne.Benoit@ens-lyon.fr

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

CR02 - 2016/2017
Outline

1. Faults and failures
2. Checkpoint and rollback recovery
3. Probabilistic models
   - Young/Daly’s approximation
   - Exponential distributions
4. Assessing protocols at scale
   - Protocol overhead of hierarchical checkpointing
   - Accounting for message logging
   - Instantiating the model
   - Experimental results
5. In-memory checkpointing
   - Double checkpointing algorithm
Outline

1. Faults and failures
2. Checkpoint and rollback recovery
3. Probabilistic models
4. Assessing protocols at scale
5. In-memory checkpointing
Outline

1. Faults and failures
2. Checkpoint and rollback recovery
3. Probabilistic models
4. Assessing protocols at scale
5. In-memory checkpointing
Outline

1. Faults and failures
2. Checkpoint and rollback recovery
3. Probabilistic models
   - Young/Daly’s approximation
   - Exponential distributions
4. Assessing protocols at scale
5. In-memory checkpointing
Outline

1. Faults and failures
2. Checkpoint and rollback recovery
3. Probabilistic models
   - Young/Daly’s approximation
   - Exponential distributions
4. Assessing protocols at scale
5. In-memory checkpointing
Checkpointing cost

Blocking model: while a checkpoint is taken, no computation can be performed
Total waste

\[
\text{WASTE} = \frac{\text{TIME}_{\text{final}} - \text{TIME}_{\text{base}}}{\text{TIME}_{\text{final}}}
\]

\[1 - \text{WASTE} = (1 - \text{WASTE}[\text{FF}]) (1 - \text{WASTE}[\text{fail}])\]

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

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

\[
\text{WASTE} = \frac{u}{T} + v + wT
\]

\[
u = C \left(1 - \frac{D + R}{\mu}\right) \quad \quad \quad \quad \quad v = \frac{D + R - C/2}{\mu} \quad \quad \quad \quad \quad w = \frac{1}{2\mu}
\]

\[
\text{WASTE minimized for } T = \sqrt{\frac{u}{w}}
\]

\[
T = \sqrt{2(\mu - (D + R))C}
\]
Comparison with Young/Daly

\[(1 - \text{WASTE}[\text{fail}]) \times \text{TIME}_{\text{final}} = \text{TIME}_{\text{FF}}\]
\[\Rightarrow T = \sqrt{2(\mu - (D + R))C}\]

Daly: \[\text{TIME}_{\text{final}} = (1 + \text{WASTE}[\text{fail}]) \times \text{TIME}_{\text{FF}}\]
\[\Rightarrow T = \sqrt{2(\mu + (D + R))C + C}\]

Young: \[\text{TIME}_{\text{final}} = (1 + \text{WASTE}[\text{fail}]) \times \text{TIME}_{\text{FF}}\text{ and } D = R = 0\]
\[\Rightarrow T = \sqrt{2\mu C + C}\]
Validity of the approach (1/3)

Technicalities

- \( \mathbb{E}(N_{\text{faults}}) = \frac{\text{TIME}_{\text{final}}}{\mu} \) and \( \mathbb{E}(T_{\text{lost}}) = D + R + \frac{T}{2} \)
  but expectation of product is not product of expectations (not independent RVs here)

- Enforce \( C \leq T \) to get \( \text{WASTE}[FF] \leq 1 \)

- Enforce \( D + R \leq \mu \) and bound \( T \) to get \( \text{WASTE}[\text{fail}] \leq 1 \)
  but \( \mu = \frac{\mu_{\text{ind}}}{p} \) too small for large \( p \), regardless of \( \mu_{\text{ind}} \)
Validity of the approach (2/3)

Several failures within same period?

- **WASTE[fail]** accurate only when two or more faults do not take place within same period

- Cap period: $T \leq \gamma \mu$, where $\gamma$ is some tuning parameter
  - Poisson process of parameter $\theta = \frac{T}{\mu}$
  - Probability of having $k \geq 0$ failures: $P(X = k) = \frac{\theta^k}{k!} e^{-\theta}$
  - Probability of having two or more failures:
    $\pi = P(X \geq 2) = 1 - (P(X = 0) + P(X = 1)) = 1 - (1 + \theta) e^{-\theta}$
  - $\gamma = 0.27 \Rightarrow \pi \leq 0.03$
    $\Rightarrow$ overlapping faults for only 3% of checkpointing segments
Validity of the approach (3/3)

- Enforce $T \leq \gamma \mu$, $C \leq \gamma \mu$, and $D + R \leq \gamma \mu$

- Optimal period $\sqrt{2(\mu - (D + R))} C$ may not belong to admissible interval $[C, \gamma \mu]$

- Waste is then minimized for one of the bounds of this admissible interval (by convexity)
Wrap up

- Capping periods, and enforcing a lower bound on MTBF
  ⇒ mandatory for mathematical rigor 😞

- Not needed for practical purposes 😊
  - actual job execution uses optimal value
  - account for multiple faults by re-executing work until success

- Approach surprisingly robust 😊
Lesson learnt for fail-stop failures

(Not so) Secret data
• Tsubame 2: 962 failures during last 18 months so \( \mu = 13 \) hrs
• Blue Waters: 2-3 node failures per day
• Titan: a few failures per day
• Tianhe 2: wouldn’t say

\[
T_{opt} = \sqrt{2\mu C} \quad \Rightarrow \quad \text{WASTE}[opt] \approx \sqrt{\frac{2C}{\mu}}
\]

<table>
<thead>
<tr>
<th>Scale</th>
<th>( C )</th>
<th>( \mu )</th>
<th>\text{WASTE}[opt]</th>
</tr>
</thead>
<tbody>
<tr>
<td>Petascale</td>
<td>20 min</td>
<td>24 hrs</td>
<td>17%</td>
</tr>
<tr>
<td>Scale by 10</td>
<td>20 min</td>
<td>2.4 hrs</td>
<td>53%</td>
</tr>
<tr>
<td>Scale by 100</td>
<td>20 min</td>
<td>0.24 hrs</td>
<td>100%</td>
</tr>
</tbody>
</table>

Anne.Benoit@ens-lyon.fr  CR02  Fault tolerance (2)
Lesson learnt for fail-stop failures

(Not so) Secret data
- Tsubame 2: 962 failures during last 18 months so $\mu = 13$ hrs
- Blue Waters: 2-3 node failures per day
- Titan: a few failures per day
- Tianhe: wouldn't say

$t_{\text{opt}} = \sqrt{2\mu} \Rightarrow W_{\text{opt}} \approx \sqrt{2\mu}$

Petascale: $C = 20$ min $\mu = 24$ hrs $\Rightarrow W_{\text{opt}} = 17\%$
Scale by 10: $C = 20$ min $\mu = 2.4$ hrs $\Rightarrow W_{\text{opt}} = 53\%$
Scale by 100: $C = 20$ min $\mu = 0.24$ hrs $\Rightarrow W_{\text{opt}} = 100\%$

Exascale $\neq$ Petascale $\times 1000$
Need more reliable components
Need to checkpoint faster
Lesson learnt for fail-stop failures

(Not so) Secret data
- Tsubame 2: 962 failures during last 18 months so $\mu = 13$ hrs
- Blue Waters: 2-3 node failures per day
- Titan: a few failures per day
- Tianhe 2: wouldn’t say

Silent errors:
detection latency $\Rightarrow$ additional problems

Petascale: $C = 20$ min $\mu = 24$ hrs $\Rightarrow$ WASTE[opt] = 17%
Scale by 10: $C = 20$ min $\mu = 2.4$ hrs $\Rightarrow$ WASTE[opt] = 53%
Scale by 100: $C = 20$ min $\mu = 0.24$ hrs $\Rightarrow$ WASTE[opt] = 100%
Outline

1. Faults and failures
2. Checkpoint and rollback recovery
3. Probabilistic models
   - Young/Daly’s approximation
   - Exponential distributions
4. Assessing protocols at scale
5. In-memory checkpointing
Exponential failure distribution

1. Expected execution time for a single chunk
2. Expected execution time for a sequential job
3. Expected execution time for a parallel job
Expected execution time for a single chunk

Compute the expected time $\mathbb{E}(T(W, C, D, R, \lambda))$ to execute a work of duration $W$ followed by a checkpoint of duration $C$.

**Recursive Approach**

$\mathbb{E}(T(W)) =$
Expected execution time for a single chunk

Compute the expected time $\mathbb{E}(T(W, C, D, R, \lambda))$ to execute a work of duration $W$ followed by a checkpoint of duration $C$.

**Recursive Approach**

\[
\mathbb{E}(T(W)) = P_{\text{suc}}(W + C)(W + C)
\]
Expected execution time for a single chunk

Compute the expected time $\mathbb{E}(T(W, C, D, R, \lambda))$ to execute a work of duration $W$ followed by a checkpoint of duration $C$.

**Recursive Approach**

\[
\mathbb{E}(T(W)) = P_{\text{succ}}(W + C)(W + C)
\]
Expected execution time for a single chunk

Compute the expected time $\mathbb{E}(T(W, C, D, R, \lambda))$ to execute a work of duration $W$ followed by a checkpoint of duration $C$.

Recursive Approach

$$\mathbb{E}(T(W)) = \mathcal{P}_{\text{succ}}(W + C)(W + C)$$
Expected execution time for a single chunk

Compute the expected time $\mathbb{E}(T(W, C, D, R, \lambda))$ to execute a work of duration $W$ followed by a checkpoint of duration $C$.

**Recursive Approach**

$$
\mathbb{E}(T(W)) = \mathcal{P}_{\text{succ}}(W + C)(W + C) + (1 - \mathcal{P}_{\text{succ}}(W + C))(\mathbb{E}(T_{\text{lost}}(W + C)) + \mathbb{E}(T_{\text{rec}}) + \mathbb{E}(T(W)))
$$
Compute the expected time $\mathbb{E}(T(W, C, D, R, \lambda))$ to execute a work of duration $W$ followed by a checkpoint of duration $C$.

**Recursive Approach**

$$\mathbb{E}(T(W)) = P_{\text{succ}}(W + C)(W + C) + (1 - P_{\text{succ}}(W + C)) (\mathbb{E}(T_{\text{lost}}(W + C)) + \mathbb{E}(T_{\text{rec}}) + \mathbb{E}(T(W)))$$

Probability of failure
Expected execution time for a single chunk

Compute the expected time $\mathbb{E}(T(W, C, D, R, \lambda))$ to execute a work of duration $W$ followed by a checkpoint of duration $C$.

**Recursive Approach**

\[
\mathbb{E}(T(W)) = \mathcal{P}_{\text{succ}}(W + C)(W + C) \\
+ (1 - \mathcal{P}_{\text{succ}}(W + C)) \left( \mathbb{E}(T_{\text{lost}}(W + C)) + \mathbb{E}(T_{\text{rec}}) + \mathbb{E}(T(W)) \right)
\]

Time elapsed before failure stroke
Expected execution time for a single chunk

Compute the expected time $\mathbb{E}(T(W, C, D, R, \lambda))$ to execute a work of duration $W$ followed by a checkpoint of duration $C$.

Recursive Approach

\[
\mathbb{E}(T(W)) = \mathcal{P}_{\text{succ}}(W + C)(W + C) + (1 - \mathcal{P}_{\text{succ}}(W + C)) (\mathbb{E}(T_{\text{lost}}(W + C)) + \mathbb{E}(T_{\text{rec}}) + \mathbb{E}(T(W)))
\]

Time needed to perform downtime and recovery
Expected execution time for a single chunk

Compute the expected time $\mathbb{E}(T(W, C, D, R, \lambda))$ to execute a work of duration $W$ followed by a checkpoint of duration $C$.

**Recursive Approach**

$$\mathbb{E}(T(W)) = \mathcal{P}_{\text{succ}}(W + C)(W + C) + (1 - \mathcal{P}_{\text{succ}}(W + C))(\mathbb{E}(T_{\text{lost}}(W + C)) + \mathbb{E}(T_{\text{rec}}) + \mathbb{E}(T(W)))$$

Time needed to compute $W$ anew
Computation of $\mathbb{E}(T(W, C, D, R, \lambda))$

\[
\mathbb{P}_{\text{succ}}(W + C)(W + C) \\
\mathbb{E}(T(W)) = + \\
(1 - \mathbb{P}_{\text{succ}}(W + C))(\mathbb{E}(T_{\text{lost}}(W + C)) + \mathbb{E}(T_{\text{rec}}) + \mathbb{E}(T(W)))
\]

- $\mathbb{P}_{\text{succ}}(W + C) = e^{-\lambda(W+C)}$
- $\mathbb{E}(T_{\text{lost}}(W + C)) = \int_{0}^{\infty} x \mathbb{P}(X = x | X < W + C) dx = \frac{1}{\lambda} - \frac{W+C}{e^{\lambda(W+C)} - 1}$
- $\mathbb{E}(T_{\text{rec}}) = e^{-\lambda R}(D+R) + (1 - e^{-\lambda R})(D + \mathbb{E}(T_{\text{lost}}(R)) + \mathbb{E}(T_{\text{rec}}))$

\[
\mathbb{E}(T(W, C, D, R, \lambda)) = e^{\lambda R} \left( \frac{1}{\lambda} + D \right) \left( e^{\lambda(W+C)} - 1 \right)
\]
Checkpoints

Checkpoiniting a sequential job

\[ \mathbb{E}(T(W)) = e^{\lambda R} \left( \frac{1}{\lambda} + D \right) \left( \sum_{i=1}^{K} e^{\lambda (W_i+C)} - 1 \right) \]

- Optimal strategy uses same-size chunks (convexity)
- \[ K_0 = \frac{\lambda W}{1 + \mathbb{L}(-e^{-\lambda C-1})} \] where \( \mathbb{L}(z)e^{\mathbb{L}(z)} = z \) (Lambert function)
- Optimal number of chunks \( K^* \) is \( \max(1, \lfloor K_0 \rfloor) \) or \( \lceil K_0 \rceil \)

\[ \mathbb{E}_{opt}(T(W)) = K^* \left( e^{\lambda R} \left( \frac{1}{\lambda} + D \right) \right) \left( e^{\lambda \left( \frac{W}{K^*} + C \right)} - 1 \right) \]

- Can also use Daly’s second-order approximation
Checkpointing a parallel job

- $p$ processors $\Rightarrow$ distribution $\text{Exp}(\lambda_p)$, where $\lambda_p = p\lambda$
- Use $W(p)$, $C(p)$, $R(p)$ in $E_{\text{opt}}(T(W))$ for a distribution $\text{Exp}(\lambda_p = p\lambda)$
- Job types
  - Perfectly parallel jobs: $W(p) = W/p$
  - Generic parallel jobs: $W(p) = W/p + \delta W$
  - Numerical kernels: $W(p) = W/p + \delta W^{2/3}/\sqrt{p}$
- Checkpoint overhead
  - Proportional overhead: $C(p) = R(p) = \delta V/p = C/p$
    (bandwidth of processor network card/link is I/O bottleneck)
  - Constant overhead: $C(p) = R(p) = \delta V = C$
    (bandwidth to/from resilient storage system is I/O bottleneck)
Weibull failure distribution

- No optimality result known
- Heuristic: maximize expected work before next failure
- Dynamic programming algorithms
  - Use a time quantum
  - Trim history of previous failures
Outline

1. Faults and failures
2. Checkpoint and rollback recovery
3. Probabilistic models
4. Assessing protocols at scale
   - Protocol overhead of hierarchical checkpointing
   - Accounting for message logging
   - Instantiating the model
   - Experimental results
5. In-memory checkpointing
Hierarchical checkpointing

- Clusters of processes
- Coordinated checkpointing protocol within clusters
- Message logging protocols between clusters
- Only processors from failed group need to roll back

⚠️ Need to log inter-group messages
- Slowdowns failure-free execution
- Increases checkpoint size/time

😊 Faster re-execution with logged messages
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-group 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
Blocking vs. non-blocking

**Blocking model:** checkpointing blocks all computations
Blocking vs. non-blocking

Non-blocking model: checkpointing has no impact on computations (e.g., first copy state to RAM, then copy RAM to disk)
General model: checkpointing slows computations down: during a checkpoint of duration $C$, the same amount of computation is done as during a time $\alpha C$ without checkpointing ($0 \leq \alpha \leq 1$)
Waste in fault-free execution

Time elapsed since last checkpoint: $T$

Amount of computations executed: $\text{WORK} = (T - C) + \alpha C$

$\text{WASTE}[^{FF}] = \frac{T - \text{WORK}}{T}$
Waste due to failures

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

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

P₀
P₁
P₂
P₃
Waste due to failures

![Diagram showing time spent working, time spent checkpointing, and time spent working with slowdown for processes $P_0$, $P_1$, $P_2$, and $P_3$.]
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

Diagram showing:
- $P_0$, $P_1$, $P_2$, $P_3$
- Time spent working, checkpointing, working with slowdown, downtime

Anne.Benoit@ens-lyon.fr
CR02 Fault tolerance (2)
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-execution 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
Total waste

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

Optimal period \[ T_{\text{opt}} = \sqrt{2(1-\alpha)(\mu - (D + R + \alpha C))C} \]
Outline

1. Faults and failures
2. Checkpoint and rollback recovery
3. Probabilistic models
4. Assessing protocols at scale
   - Protocol overhead of hierarchical checkpointing
   - Accounting for message logging
   - Instantiating the model
   - Experimental results
5. In-memory checkpointing
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

$$\alpha(G-g+1)C$$

$$T_{lost}$$

$$D$$

$$R$$

$$T_{lost}$$

$$T - G.C - T_{lost}$$
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

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

\[ \text{WASTE}[FF] = \frac{T - \text{WORK}}{T} \]
\[ \text{where WORK} = T - (1 - \alpha)GC(q) \]
Failure during computation phase

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

G1, G2, G3, G4, G5

Anne.Benoit@ens-lyon.fr CR02 Fault tolerance (2)
Failure during computation phase

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

G_1  
G_2  
G_3  
G_4  
G_5  

Time
Failure during computation phase

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

Faults Checkpoints Proba models Hierarchical Buddy

Anne.Benoit@ens-lyon.fr CR02 Fault tolerance (2)
Failure during computation phase

Tightly-coupled model: while one group is in downtime, none can work
Faults Checkpoints Proba models Hierarchical Buddy

Failure during computation phase

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

Time

$G_1$ $G_2$ $G_3$ $G_4$ $G_5$

Tightly-coupled model: while one group is in recovery, none can work

Anne.Benoit@ens-lyon.fr

CR02 Fault tolerance (2)
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 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
Faults Checkpoints Proba models Hierarchical Buddy

Failure during computation phase

Redo work done in computation phase and that was destroyed by the failure
Faults Checkpoints Proba models Hierarchical Buddy

Failure during computation phase

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

Time

\( G_1 \)
\( G_2 \)
\( G_g \)
\( G_4 \)
\( G_5 \)

\( T_{lost} \)

\( T_{lost} \)

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

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

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

Average approximated RE-EXEC:

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

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

G1 | G2 | G3 | G4 | G5

Time
Failure during checkpointing phase

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

$T - G.C$
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
Failure during checkpointing phase: before checkpoint

<table>
<thead>
<tr>
<th>Time spent working</th>
<th>Time spent checkpointing</th>
<th>Time spent working with slowdown</th>
</tr>
</thead>
<tbody>
<tr>
<td>Downtime</td>
<td>Recovery time</td>
<td>Re-executing slowed-down work</td>
</tr>
</tbody>
</table>

- $G_1$
- $G_2$
- $G_3$
- $G_g$
- $G_5$
Failure during checkpointing phase: during checkpointing

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

G1, G2, G3, Gg, G5
Failure during checkpointing phase: after checkpointing
Average waste for failures during checkpointing phase

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

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

$$\frac{1}{G}((g-1) \cdot \text{RE-EXEC}_{before\_ckpt} + 1 \cdot \text{RE-EXEC}_{during\_ckpt} + (G-g) \cdot \text{RE-EXEC}_{after\_ckpt})$$
Total waste

\[ \text{WASTE}[FF] = \frac{T - \text{WORK}}{T} \text{ with } \text{WORK} = T - (1 - \alpha)GC(q) \]

\[ \text{WASTE}[\text{fail}] = \frac{1}{\mu} \left( D(q) + R(q) + \text{RE-EXEC} \right) \text{ with} \]

\[ \text{RE-EXEC} = \frac{T - GC(q)}{T} \text{RE-EXEC}_{\text{comp}} + \frac{GC(q)}{T} \text{RE-EXEC}_{\text{ckpt}} \]

\[ \text{WASTE} = \text{WASTE}[FF] + \text{WASTE}[\text{fail}] - \text{WASTE}[FF]\text{WASTE}[\text{fail}] \]

Minimize \text{WASTE} subject to:

- \( GC(q) \leq T \) (by construction)
- Gets complicated! Use computer algebra software 😞
Outline

1. Faults and failures
2. Checkpoint and rollback recovery
3. Probabilistic models
4. Assessing protocols at scale
   - Protocol overhead of hierarchical checkpointing
   - Accounting for message logging
   - Instantiating the model
   - Experimental results
5. In-memory checkpointing
Accounting for message logging: Impact on work

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

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

\[
\text{WASTE}[FF] = \frac{T - \lambda \text{WORK}}{T}
\]

\[
\text{WASTE}[fail] = \frac{1}{\mu} \left( D(q) + R(q) + \frac{\text{RE-EXEC}}{\rho} \right)
\]
Accounting for message logging: Impact on checkpoint size

- Inter-group 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

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

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

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

1. Faults and failures
2. Checkpoint and rollback recovery
3. Probabilistic models
4. **Assessing protocols at scale**
   - Protocol overhead of hierarchical checkpointing
   - Accounting for message logging
   - **Instantiating the model**
   - Experimental results
5. In-memory checkpointing
Three case studies

**Coord-IO**
Coordinated approach: \( C = C_{\text{Mem}} = \frac{\text{Mem}}{b_{io}} \)
where Mem is the memory footprint of the application

**Hierarch-IO**
Several (large) groups, *I/O-saturated*
⇒ groups checkpoint sequentially

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

**Hierarch-Port**
Very large number of smaller groups, *port-saturated*
⇒ some groups checkpoint in parallel
Groups of \( q_{\text{min}} \) processors, where \( q_{\text{min}} b_{port} \geq b_{io} \)
Three applications

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

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

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

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

Each process sends a block to its 4 neighbors

**Hierarch-IO:**
- 1 group = 1 grid row
- 2 out of the 4 messages are logged

$$\beta = \frac{\text{Logged}_\text{Msg}}{C_0(q)\text{WORK}} = \frac{2pb}{pb^2(9b^2/sp)} = \frac{2sp}{9b^3}$$

**Hierarch-Port:**
- $\beta$ doubles
Three applications: Matrix product

- Three matrices involved: \( \text{Mem} = 3n^2 \), \( C_0(q) = 3pb^2 \)
- Cannon’s algorithm: \( p \) steps to compute a product
- \( \text{Work} = \frac{2b^3}{s_p}, b = n/p \)

Hierarch-IO:

- 1 group = 1 grid row
- only vertical messages are logged: \( pb^2 \)
- \( \beta = \frac{pb^2}{3pb^2(2b^3/s_p)} = \frac{s_p}{6b^3} \)

Hierarch-Port:

- \( \beta \) doubles
Three applications: 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 \( \text{WORK} = \frac{27b^3}{sp} \)

Three hierarchical variants

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

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

3. **Hierarch-Port**: groups of size \( q_{\text{min}} \):
   \[ \beta = \frac{6sp}{27b^3} \]
3D-stencil illustration

- 3D-Plane: Vertical messages are logged
- 3D-Line: Twice as many messages are logged
3D-stencil illustration

- 3D-Plane: Vertical messages are logged
- 3D-Line: Twice as many messages are logged
3D-stencil illustration

- **3D-Plane**: Vertical messages are logged.
- **3D-Line**: Twice as many messages are logged.
## 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</th>
<th>I/O Network Bandwidth (b_{io})</th>
<th>I/O Bandwidth (b_{port})</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</td>
<td>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</td>
<td>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</td>
<td>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</td>
<td>400GB/s</td>
</tr>
</tbody>
</table>

### Faults Checkpoints

<table>
<thead>
<tr>
<th>Name</th>
<th>Scenario</th>
<th>$C(q)$</th>
<th>$\beta$ for 2D-Stencil</th>
<th>$\beta$ for Matrix-Product</th>
</tr>
</thead>
<tbody>
<tr>
<td>Titan</td>
<td>COORD-IO</td>
<td>1 (2,048s)</td>
<td>0.00001098</td>
<td>0.0004280</td>
</tr>
<tr>
<td></td>
<td>HIERARCH-IO</td>
<td>136 (15s)</td>
<td>0.0002196</td>
<td>0.0008561</td>
</tr>
<tr>
<td></td>
<td>HIERARCH-PORT</td>
<td>1,246 (1.6s)</td>
<td>/</td>
<td>/</td>
</tr>
<tr>
<td>K-Computer</td>
<td>COORD-IO</td>
<td>1 (14,688s)</td>
<td>0.00002858</td>
<td>0.001113</td>
</tr>
<tr>
<td></td>
<td>HIERARCH-IO</td>
<td>296 (50s)</td>
<td>0.0005716</td>
<td>0.002227</td>
</tr>
<tr>
<td></td>
<td>HIERARCH-PORT</td>
<td>17,626 (0.83s)</td>
<td>/</td>
<td>/</td>
</tr>
<tr>
<td>Exascale-Slim</td>
<td>COORD-IO</td>
<td>1 (64,000s)</td>
<td>0.0002599</td>
<td>0.001013</td>
</tr>
<tr>
<td></td>
<td>HIERARCH-IO</td>
<td>1,000 (64s)</td>
<td>0.0005199</td>
<td>0.002026</td>
</tr>
<tr>
<td></td>
<td>HIERARCH-PORT</td>
<td>200,000 (0.32s)</td>
<td>/</td>
<td>/</td>
</tr>
<tr>
<td>Exascale-Fat</td>
<td>COORD-IO</td>
<td>1 (64,000s)</td>
<td>0.00008220</td>
<td>0.0003203</td>
</tr>
<tr>
<td></td>
<td>HIERARCH-IO</td>
<td>316 (217s)</td>
<td>0.00016440</td>
<td>0.0006407</td>
</tr>
<tr>
<td></td>
<td>HIERARCH-PORT</td>
<td>33,3333 (1.92s)</td>
<td>/</td>
<td>/</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. Faults and failures
2. Checkpoint and rollback recovery
3. Probabilistic models
4. Assessing protocols at scale
   - Protocol overhead of hierarchical checkpointing
   - Accounting for message logging
   - Instantiating the model
   - Experimental results
5. In-memory checkpointing
Simulation parameters

- Failure distribution: Weibull, $k = 0.7$
- Failure free execution on each process: 4 days
- Time-out: 1 year
- No assumption on failures
- $\alpha = 0.3$, $\lambda = 0.98$, $\rho = 1.5$
- Each point: average over 20 randomly generated instances
- Computed period and best period:
  - Generate 480 periods in the neighborhood of the period from the model
  - Numerically evaluate the best one through simulations
Faults Checkpoints Proba models Hierarchical Buddy

Plotting formulas – Platform: Titan

Stencil 2D

Matrix product

Stencil 3D

Waste as a function of processor MTBF $\mu_{ind}$
Faults Checkpoints Proba models Hierarchical Buddy

Platform: K-Computer

Stencil 2D  Matrix product  Stencil 3D

Waste as a function of processor MTBF $\mu_{ind}$

Anne.Benoit@ens-lyon.fr CR02 Fault tolerance (2)
WASTE = 1 for all scenarios!!!
WASTE = 0 for all scenarios!!!

Goodbye Exascale?!
## Checkpoint time

<table>
<thead>
<tr>
<th>Name</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>K-Computer</td>
<td>14,688s</td>
</tr>
<tr>
<td>Exascale-Slim</td>
<td>64,000s</td>
</tr>
<tr>
<td>Exascale-Fat</td>
<td>64,000s</td>
</tr>
</tbody>
</table>

- Large time to dump the memory
- Using 1% C
  - faster I/O and storage (two-level checkpoint, SSD, ...)
  - smaller amount of memory written
- Comparing with 0.1% C for exascale platforms
Platform: KComputer with $C = C/100$

- Solid line: Computed period
- Dotted line: Best period

Waste as a function of processor MTBF $\mu_{ind}$, $C = C/100$
Platform: Exascale with $C = C/100$

- Solid line: Computed period
- Dotted line: Best period

Waste as a function of processor MTBF $\mu_{ind}$, $C = C/100$
Checkpoint impact: Exascale Slim

- Solid line: Computed period
- Dotted line: Best period

\[ C = \frac{C}{100} \]

\[ C = \frac{C}{1000} \]

Waste as a function of processor MTBF \( \mu \) with checkpoint variation
Checkpoint impact: Exascale Fat

- Solid line: Computed period
- Dotted line: Best period

C = C/100

1 2 3 4 5 7.5 10 15 20 25 30 35 50 75 100

C = C/1000

1 2 3 4 5 7.5 10 15 20 25 30 35 50 75 100

2D-Stencil

Matrix Product

3D-Stencil

Waste as a function of processor MTBF $\mu$ with checkpoint variation

Anne.Benoit@ens-lyon.fr
Conclusion

- Hierarchical protocols very sensitive to message logging: direct relationship between $\beta$ and the observed waste
- Hierarchical protocols better for small MTBFs: more suitable for failure-prone platforms
- Struggle when communication intensity increases (3D-stencil), but limited waste in all other cases
- The faster the checkpointing time, the smaller the waste
- Exascale-Fat better than Exascale-Slim: fewer processors, hence larger MTBF!
- Simulations with random trace of errors: the model computes near-optimal checkpointing periods
- What could we further add: (partial) replication, prediction, energy
Outline

1. Faults and failures
2. Checkpoint and rollback recovery
3. Probabilistic models
4. Assessing protocols at scale
5. In-memory checkpointing
   - Double checkpointing algorithm
Motivation

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

- Stable storage: high cost

- Distributed in-memory storage:
  - Store checkpoints in local memory ⇒ no centralized storage 😊 Much better scalability
  - Replicate checkpoints ⇒ application survives single failure 😞 Still, risk of fatal failure in some (unlikely) scenarios
Outline

1. Faults and failures
2. Checkpoint and rollback recovery
3. Probabilistic models
4. Assessing protocols at scale
5. In-memory checkpointing
   - Double checkpointing algorithm
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