An overview of fault-tolerant techniques for HPC

Thomas Hérault\textsuperscript{1} & Yves Robert\textsuperscript{1,2}

1 – University of Tennessee Knoxville
2 – ENS Lyon & Institut Universitaire de France

herault@icl.utk.edu | yves.robert@ens-lyon.fr
http://graal.ens-lyon.fr/~yrobert/sc13tutorial.pdf

SC'2013 Tutorial
Thanks

INRIA & ENS Lyon
- Anne Benoit
- Frédéric Vivien
- PhD students (Guillaume Aupy, Dounia Zaidouni)

UT Knoxville
- George Bosilca
- Aurélien Bouteiller
- Jack Dongarra

Others
- Franck Cappello, Argonne and UIUC-Inria joint lab
- Henri Casanova, Univ. Hawaiʻi
- Amina Guermouche, UIUC-Inria joint lab
Outline

1. Introduction (15mn)
   - Large-scale computing platforms
   - Faults and failures

2. General-purpose fault-tolerance techniques (30mn)
   - Replication
   - Process Checkpointing
   - Coordinated Checkpointing
   - Uncoordinated checkpointing
Outline

1. Introduction (15mn)
   - Large-scale computing platforms
   - Faults and failures

2. General-purpose fault-tolerance techniques (30mn)
   - Replication
   - Process Checkpointing
   - Coordinated Checkpointing
   - Uncoordinated checkpointing
Outline

1. Introduction (15mn)
   - Large-scale computing platforms
   - Faults and failures

2. General-purpose fault-tolerance techniques (30mn)
   - Replication
   - Process Checkpointing
   - Coordinated Checkpointing
   - Uncoordinated checkpointing
Potential System Architecture with a cap of $200M and 20MW

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>System peak</td>
<td>10.5 Pflop/s</td>
<td>1 Eflop/s</td>
<td>O(100)</td>
</tr>
<tr>
<td>Power</td>
<td>12.7 MW</td>
<td>~20 MW</td>
<td></td>
</tr>
<tr>
<td>System memory</td>
<td>1.6 PB</td>
<td>32 - 64 PB</td>
<td>O(10)</td>
</tr>
<tr>
<td>Node performance</td>
<td>128 GF</td>
<td>1,2 or 15TF</td>
<td>O(10) – O(100)</td>
</tr>
<tr>
<td>Node memory BW</td>
<td>64 GB/s</td>
<td>2 - 4TB/s</td>
<td>O(100)</td>
</tr>
<tr>
<td>Node concurrency</td>
<td>8</td>
<td>O(1k) or 10k</td>
<td>O(100) – O(1000)</td>
</tr>
<tr>
<td>Total Node Interconnect BW</td>
<td>20 GB/s</td>
<td>200-400GB/s</td>
<td>O(10)</td>
</tr>
<tr>
<td>System size (nodes)</td>
<td>88,124</td>
<td>O(100,000) or O(1M)</td>
<td>O(10) – O(100)</td>
</tr>
<tr>
<td>Total concurrency</td>
<td>705,024</td>
<td>O(billion)</td>
<td>O(1,000)</td>
</tr>
<tr>
<td>MTTI</td>
<td>days</td>
<td>O(1 day)</td>
<td>- O(10)</td>
</tr>
</tbody>
</table>
### Toward Exascale Computing (My Roadmap)

Based on proposed DOE roadmap with MTTI adjusted to scale linearly

<table>
<thead>
<tr>
<th>Systems</th>
<th>2009</th>
<th>2011</th>
<th>2015</th>
<th>2018</th>
</tr>
</thead>
<tbody>
<tr>
<td>System peak</td>
<td>2 Peta</td>
<td>20 Peta</td>
<td>100-200 Peta</td>
<td>1 Exa</td>
</tr>
<tr>
<td>System memory</td>
<td>0.3 PB</td>
<td>1.6 PB</td>
<td>5 PB</td>
<td>10 PB</td>
</tr>
<tr>
<td>Node performance</td>
<td>125 GF</td>
<td>200GF</td>
<td>200-400 GF</td>
<td>1-10TF</td>
</tr>
<tr>
<td>Node memory BW</td>
<td>25 GB/s</td>
<td>40 GB/s</td>
<td>100 GB/s</td>
<td>200-400 GB/s</td>
</tr>
<tr>
<td>Node concurrency</td>
<td>12</td>
<td>32</td>
<td>O(100)</td>
<td>O(1000)</td>
</tr>
<tr>
<td>Interconnect BW</td>
<td>1.5 GB/s</td>
<td>22 GB/s</td>
<td>25 GB/s</td>
<td>50 GB/s</td>
</tr>
<tr>
<td>System size (nodes)</td>
<td>18,700</td>
<td>100,000</td>
<td>500,000</td>
<td>O(million)</td>
</tr>
<tr>
<td>Total concurrency</td>
<td>225,000</td>
<td>3,200,000</td>
<td>O(50,000,000)</td>
<td>O(billion)</td>
</tr>
<tr>
<td>Storage</td>
<td>15 PB</td>
<td>30 PB</td>
<td>150 PB</td>
<td>300 PB</td>
</tr>
<tr>
<td>IO</td>
<td>0.2 TB/s</td>
<td>2 TB/s</td>
<td>10 TB/s</td>
<td>20 TB/s</td>
</tr>
<tr>
<td>MTTI</td>
<td>4 days</td>
<td>19 h 4 min</td>
<td>3 h 52 min</td>
<td>1 h 56 min</td>
</tr>
<tr>
<td>Power</td>
<td>6 MW</td>
<td>~10MW</td>
<td>~10 MW</td>
<td>~20 MW</td>
</tr>
</tbody>
</table>
Exascale platforms

- **Hierarchical**
  - 10^5 or 10^6 nodes
  - Each node equipped with 10^4 or 10^3 cores

- **Failure-prone**

<table>
<thead>
<tr>
<th>MTBF – one node of 10^6 nodes</th>
<th>1 year 30sec</th>
<th>10 years 5mn</th>
<th>120 years 1h</th>
</tr>
</thead>
</table>

More nodes ⇒ Shorter MTBF (Mean Time Between Failures)
Exascale platforms

- Hierarchical
  - $10^5$ or $10^6$ nodes
  - Each node equipped with $10^4$ or $10^3$ cores

- Failure-prone

<table>
<thead>
<tr>
<th>MTBF – one node</th>
<th>MTBF – platform of $10^6$ nodes</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 year 30sec</td>
<td>1 year 5mn</td>
</tr>
<tr>
<td>10 years 5mn</td>
<td>120 years 1h</td>
</tr>
</tbody>
</table>

More nodes \(\neq\) Petascale \(\times 1000\)
Even for today’s platforms (courtesy F. Cappello)

Fault tolerance becomes critical at Petascale (MTTI <= 1 day)
Poor fault tolerance design may lead to huge overhead

Today, 20% or more of the computing capacity in a large high-performance computing system is wasted due to failures and recoveries.

Dr. E.N. (Mootaz) Elnozahyet al. System Resilience at Extreme Scale, DARPA
Even for today’s platforms (courtesy F. Cappello)

Classic approach for FT: Checkpoint-Restart

Typical “Balanced Architecture” for PetaScale Computers

Compute nodes
Total memory: 100-200 TB
Network(s)

40 to 200 GB/s
Parallel file system (1 to 2 PB)

I/O nodes

Without optimization, Checkpoint-Restart needs about 1h! (~30 minutes each)

<table>
<thead>
<tr>
<th>Systems</th>
<th>Perf.</th>
<th>Ckpt time</th>
<th>Source</th>
</tr>
</thead>
<tbody>
<tr>
<td>RoadRunner</td>
<td>1PF</td>
<td>~20 min.</td>
<td>Panasas</td>
</tr>
<tr>
<td>LLNL BG/L</td>
<td>500 TF</td>
<td>&gt;20 min.</td>
<td>LLNL</td>
</tr>
<tr>
<td>LLNL Zeus</td>
<td>11TF</td>
<td>26 min.</td>
<td>LLNL</td>
</tr>
<tr>
<td>YYY BG/P</td>
<td>100 TF</td>
<td>~30 min.</td>
<td>YYY</td>
</tr>
</tbody>
</table>
Scenario for 2015

- Phase-Change memory
  - read bandwidth 100GB/sec
  - write bandwidth 10GB/sec
- Checkpoint size 128GB
- $C$: checkpoint save time: $C = 12\text{sec}$
- $R$: checkpoint recovery time: $R = 1.2\text{sec}$
- $D$: down/reboot time: $D = 15\text{sec}$
- $p$: total number of (multicore) nodes: $p = 2^8$ to $p = 2^{20}$
- MTBF $\mu = 1\text{ week, 1 month, 1|10|100|1000 years (per node)}$
Distribution of parallel jobs

Number of processors required by typical jobs: two-stage log-uniform distribution biased to powers of two (says Dr. Feitelson)

- Let \( p = 2^Z \) for simplicity
- Probability that a job is sequential: \( \alpha_0 = p_1 \approx 0.25 \)
- Otherwise, the job is parallel, and uses \( 2^j \) processors with identical probability

**Steady-state** utilization of whole platform:
- all processors always active
- constant proportion of jobs using any number of processors
### Platform throughput with optimal checkpointing period

<table>
<thead>
<tr>
<th>$\mu$ = 1 week</th>
<th>$p$</th>
<th>Throughput</th>
<th>$\mu$ = 1 month</th>
<th>$p$</th>
<th>Throughput</th>
<th>$\mu$ = 1 year</th>
<th>$p$</th>
<th>Throughput</th>
</tr>
</thead>
<tbody>
<tr>
<td>$2^8$</td>
<td></td>
<td>91.56%</td>
<td>$2^8$</td>
<td></td>
<td>96.04%</td>
<td>$2^8$</td>
<td></td>
<td>98.89%</td>
</tr>
<tr>
<td>$2^{11}$</td>
<td></td>
<td>73.75%</td>
<td>$2^{11}$</td>
<td></td>
<td>88.23%</td>
<td>$2^{11}$</td>
<td></td>
<td>96.80%</td>
</tr>
<tr>
<td>$2^{14}$</td>
<td></td>
<td>20.07%</td>
<td>$2^{14}$</td>
<td></td>
<td>62.28%</td>
<td>$2^{14}$</td>
<td></td>
<td>90.59%</td>
</tr>
<tr>
<td>$2^{17}$</td>
<td></td>
<td>2.51%</td>
<td>$2^{17}$</td>
<td></td>
<td>10.66%</td>
<td>$2^{17}$</td>
<td></td>
<td>70.46%</td>
</tr>
<tr>
<td>$2^{20}$</td>
<td></td>
<td>0.31%</td>
<td></td>
<td></td>
<td>1.33%</td>
<td>$2^{20}$</td>
<td></td>
<td>15.96%</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>$\mu$ = 10 years</th>
<th>$p$</th>
<th>Throughput</th>
<th>$\mu$ = 100 years</th>
<th>$p$</th>
<th>Throughput</th>
</tr>
</thead>
<tbody>
<tr>
<td>$2^8$</td>
<td></td>
<td>99.65%</td>
<td>$2^8$</td>
<td></td>
<td>99.89%</td>
</tr>
<tr>
<td>$2^{11}$</td>
<td></td>
<td>99.00%</td>
<td>$2^{11}$</td>
<td></td>
<td>99.69%</td>
</tr>
<tr>
<td>$2^{14}$</td>
<td></td>
<td>97.15%</td>
<td>$2^{14}$</td>
<td></td>
<td>99.11%</td>
</tr>
<tr>
<td>$2^{17}$</td>
<td></td>
<td>91.63%</td>
<td>$2^{17}$</td>
<td></td>
<td>97.45%</td>
</tr>
<tr>
<td>$2^{20}$</td>
<td></td>
<td>74.01%</td>
<td>$2^{20}$</td>
<td></td>
<td>92.56%</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>$\mu$ = 1000 years</th>
<th>$p$</th>
<th>Throughput</th>
</tr>
</thead>
<tbody>
<tr>
<td>$2^8$</td>
<td></td>
<td>99.97%</td>
</tr>
<tr>
<td>$2^{11}$</td>
<td></td>
<td>99.90%</td>
</tr>
<tr>
<td>$2^{14}$</td>
<td></td>
<td>99.72%</td>
</tr>
<tr>
<td>$2^{17}$</td>
<td></td>
<td>99.20%</td>
</tr>
<tr>
<td>$2^{20}$</td>
<td></td>
<td>97.73%</td>
</tr>
</tbody>
</table>
Outline

1. Introduction (15mn)
   - Large-scale computing platforms
   - Faults and failures

2. General-purpose fault-tolerance techniques (30mn)
   - Replication
   - Process Checkpointing
   - Coordinated Checkpointing
   - Uncoordinated checkpointing
Introduction (15mn)

General Purpose FT

Error sources (courtesy Franck Cappello)

Sources of failures

• Analysis of error and failure logs

• In 2005 (Ph. D. of CHARNG-DA LU): “Software halts account for the most number of outages (59-84 percent), and take the shortest time to repair (0.6-1.5 hours). Hardware problems, albeit rarer, need 6.3-100.7 hours to solve.”

• In 2007 (Garth Gibson, ICPP Keynote):

• In 2008 (Oliner and J. Stearley, DSN Conf.):

<table>
<thead>
<tr>
<th>Type</th>
<th>Raw Count</th>
<th>Raw %</th>
<th>Filtered Count</th>
<th>Filtered %</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hardware</td>
<td>174,586,516</td>
<td>98.04</td>
<td>1,999</td>
<td>18.78</td>
</tr>
<tr>
<td>Software</td>
<td>144,899</td>
<td>0.08</td>
<td>6,814</td>
<td>64.01</td>
</tr>
<tr>
<td>Indeterminate</td>
<td>3,350,044</td>
<td>1.88</td>
<td>1,832</td>
<td>17.21</td>
</tr>
</tbody>
</table>

Relative frequency of root cause by system type.

Software errors: Applications, OS bug (kernel panic), communication libs, File system error and other.
Hardware errors, Disks, processors, memory, network

Conclusion: Both Hardware and Software failures have to be considered
A few definitions

- Many types of faults: software error, hardware malfunction, memory corruption
- Many possible behaviors: silent, transient, unrecoverable
- Restrict to faults that lead to application failures
- This includes all hardware faults, and some software ones
- Will use terms fault and failure interchangeably
Failure distributions: (1) Exponential

**Exp(λ):** Exponential distribution law of parameter $\lambda$:

- Pdf: $f(t) = \lambda e^{-\lambda t}$ for $t \geq 0$
- Cdf: $F(t) = 1 - e^{-\lambda t}$
- Mean = $\frac{1}{\lambda}$
X random variable for $\text{Exp}(\lambda)$ failure inter-arrival times:

- $\mathbb{P}(X \leq t) = 1 - e^{-\lambda t}$ (by definition)
- Memoryless property: $\mathbb{P}(X \geq t + s \mid X \geq s) = \mathbb{P}(X \geq t)$
  - at any instant, time to next failure does not depend upon time elapsed since last failure
- Mean Time Between Failures (MTBF) $\mu = \mathbb{E}(X) = \frac{1}{\lambda}$
Weibull \((k, \lambda)\): Weibull distribution law of shape parameter \(k\) and scale parameter \(\lambda\):

- Pdf: \(f(t) = k \lambda (t \lambda)^{k-1} e^{-(\lambda t)^k}\) for \(t \geq 0\)
- Cdf: \(F(t) = 1 - e^{-(\lambda t)^k}\)
- Mean = \(\frac{1}{\lambda} \Gamma(1 + \frac{1}{k})\)
X random variable for $\text{Weibull}(k, \lambda)$ failure inter-arrival times:

- If $k < 1$: failure rate decreases with time
  “infant mortality”: defective items fail early
- If $k = 1$: $\text{Weibull}(1, \lambda) = \text{Exp}(\lambda)$ constant failure time
Failure distributions: with several processors

- Processor (or node): any entity subject to failures
  ⇒ approach agnostic to granularity

- If the MTBF is $\mu$ with one processor, what is its value with $p$ processors?

- Well, it depends 😐
Failure distributions: with several processors

- Processor (or node): any entity subject to failures
  ⇒ approach **agnostic to granularity**

- If the MTBF is $\mu$ with one processor, what is its value with $p$ processors?

- Well, it depends 😁
With rejuvenation

- Rebooting all $p$ processors after a failure
- Platform failure distribution
  $\Rightarrow$ minimum of $p$ IID processor distributions
- With $p$ distributions $\Exp(\lambda)$:

$$\min (\Exp(\lambda_1), \Exp(\lambda_2)) = \Exp(\lambda_1 + \lambda_2)$$

$$\mu = \frac{1}{\lambda} \Rightarrow \mu_p = \frac{\mu}{p}$$

- With $p$ distributions $\text{Weibull}(k, \lambda)$:

$$\min_{1..p} (\text{Weibull}(k, \lambda)) = \text{Weibull}(k, p^{1/k} \lambda)$$

$$\mu = \frac{1}{\lambda} \Gamma(1 + \frac{1}{k}) \Rightarrow \mu_p = \frac{\mu}{p^{1/k}}$$
Without rejuvenation (= real life)

- Rebooting only faulty processor
- Platform failure distribution
  \[ \Rightarrow \] superposition of \( p \) IID processor distributions

**Theorem:** \( \mu_p = \frac{\mu}{p} \) for arbitrary distributions
Values from the literature

- MTBF of one processor: between 1 and 125 years
- Shape parameters for Weibull: $k = 0.5$ or $k = 0.7$
- Failure trace archive from INRIA ([http://fta.inria.fr](http://fta.inria.fr))
- Computer Failure Data Repository from LANL ([http://institutes.lanl.gov/data/fdata](http://institutes.lanl.gov/data/fdata))
Does it matter?

Parallel machine ($10^6$ nodes)

- **Exp**(1/100)
- **Weibull**(0.7, 1/100)
- **Weibull**(0.5, 1/100)
Outline

1. Introduction (15mn)
   - Large-scale computing platforms
   - Faults and failures

2. General-purpose fault-tolerance techniques (30mn)
   - Replication
   - Process Checkpointing
   - Coordinated Checkpointing
   - Uncoordinated checkpointing
## Goal

- General Purpose Fault Tolerance Techniques: work despite the application behavior
- Two adversaries: Failures & Application
- Use automatically computed redundant information
  - At given instants: checkpoints
  - At any instant: replication
  - Or anything in between: checkpoint + message logging
Outline

1. Introduction (15mn)
   - Large-scale computing platforms
   - Faults and failures

2. General-purpose fault-tolerance techniques (30mn)
   - Replication
   - Process Checkpointing
   - Coordinated Checkpointing
   - Uncoordinated checkpointing
Replication

### Idea
- Each process is replicated on a resource that has small chance to be hit by the same failure as its replica.
- In case of failure, one of the replicas will continue working, while the other recovers.
- Passive Replication / Active Replication.
**Replication**

- **Challenges**
  - Passive replication: latency of state update
  - Active replication: ordering of decision → internal additional communications
Replication

Any replica can provide an answer (load balance)

Messages must be delivered in a consistent order to all replicas

Challenges

- Passive replication: latency of state update
- Active replication: ordering of decision → internal additional communications
Outline

1. Introduction (15mn)
   - Large-scale computing platforms
   - Faults and failures

2. General-purpose fault-tolerance techniques (30mn)
   - Replication
   - Process Checkpointing
   - Coordinated Checkpointing
   - Uncoordinated checkpointing
Process Checkpointing

Goal
- Save the current state of the process
- FT Protocols save a possible state of the parallel application

Techniques
- User-level checkpointing
- System-level checkpointing
- Blocking call
- Asynchronous call
User-level checkpointing

User code serializes the state of the process in a file.

- Usually small
- Portability
- Diversity of use

- Hard to implement if preemptive checkpointing is needed
- Loss of the functions call stack
  - code full of jumps
  - loss of internal library state
System-level checkpointing

- Different possible implementations: OS syscall; dynamic library; compiler assisted
- Create a serial file that can be loaded in a process image. Usually on the same architecture, same OS, same software environment.

Entirely transparent
- Preemptive (often needed for library-level checkpointing)

Lack of portability
- Large size of checkpoint (≈ memory footprint)
Blocking / Asynchronous call

**Blocking Checkpointing**

Relatively intuitive: `checkpoint(filename)`
Cost: no process activity during the whole checkpoint operation.
Can be linear in the size of memory and in the size of modified files

**Asynchronous Checkpointing**

System-level approach: make use of copy on write of `fork` syscall
User-level approach: critical sections, when needed
Storage

Remote Reliable Storage


Memory Hierarchy

- local memory
- local disk (SSD, HDD)
- remote disk
  - Scalable Checkpoint Restart Library
    http://scalablecr.sourceforge.net

Checkpoint is valid when finished on reliable storage

Distributed Memory Storage

- In-memory checkpointing
- Disk-less checkpointing
Outline

1. Introduction (15mn)
   - Large-scale computing platforms
   - Faults and failures

2. General-purpose fault-tolerance techniques (30mn)
   - Replication
   - Process Checkpointing
   - Coordinated Checkpointing
   - Uncoordinated checkpointing
Coordinated checkpointing

Definition (Missing Message)
A message is missing if in the current configuration, the sender sent, while the receiver did not receive it.
Coordinated checkpointing

Definition (Orphan Message)

A message is orphan if in the current configuration, the receiver received it, while the sender did not send it.
Coordinated Checkpointing Idea

Create a consistent view of the application

- Messages belong to a checkpoint wave or another
- All communication channels must be flushed (all2all)
Blocking Coordinated Checkpointing

- Silences the network during the checkpoint

App. Message ▶ Marker Message
Non-Blocking Coordinated Checkpointing

- Communications received after the beginning of the checkpoint and before its end are added to the receiver’s checkpoint.
- Communications inside a checkpoint are pushed back at the beginning of the queues.
Implementation

Communication Library

- Flush of communication channels
  - Conservative approach. One Message per open channel / One message per channel
- Preemptive checkpointing usually required
  - Can have a user-level checkpointing, but requires one that can be called any time

Application Level

- Flush of communication channels
  - Can be as simple as `Barrier(); Checkpoint();`
  - Or as complex as having a `quiesce();` function in all libraries
- User-level checkpointing
Coordinated Protocol Performance

- **VCL** = nonblocking coordinated protocol
- **PCL** = blocking coordinated protocol

In the figures:
- **VCL** execution time
- **PCL** execution time
- **VCL number of waves**
- **PCL number of waves**
Outline

1. Introduction (15mn)
   - Large-scale computing platforms
   - Faults and failures

2. General-purpose fault-tolerance techniques (30mn)
   - Replication
   - Process Checkpointing
   - Coordinated Checkpointing
   - Uncoordinated checkpointing
Uncoordinated Checkpointing Idea

Processes checkpoint independently
Uncoordinated Checkpointing Idea

Optimistic Protocol

- Each process $i$ keeps some checkpoints $C_i^j$
- $\forall (i_1, \ldots, i_n), \exists j_k / \{C_{i_k}^{j_k}\}$ form a consistent cut?
- Domino Effect
Piece-wise Deterministic Assumption

- Process: alternate sequence of non-deterministic choice and deterministic steps
- Translated in Message Passing:
  - Receptions / Progress test are non-deterministic
    \[
    \text{MPI\_Wait(ANY\_SOURCE), if( MPI\_Test() )<...>; else <...>}
    \]
  - Emissions / others are deterministic

herault@icl.utk.edu — yves.robert@ens-lyon.fr
Message Logging

By replaying the sequence of messages and test/probe with the same result that it obtained in the initial execution (from the last checkpoint), one can guide the execution of a process to its exact state just before the failure.
Message Logging

Message / Events

- Message = unique identifier (source, emission index, destination, reception index) + payload (content of the message)
- Probe = unique identifier (number of consecutive failed/success probes on this link)
- Event Logging: saving the unique identifier of a message, or of a probe
Message Logging

- **Payload Logging**: saving the content of a message
- **Message Logging**: saving the unique identifier and the payload of a message, saving unique identifiers of probes, saving the (local) order of events
Message Logging

Where to save the Payload?

- Almost always as Sender Based
- Local copy: less impact on performance
- More memory demanding → trade-off garbage collection algorithm
- Payload needs to be included in the checkpoints

P will never be requested again
Q might be requested if A and B rollback
Message Logging

Where to save the Events?

- Events must be saved on a reliable space
- Must avoid: loss of events ordering information, for all events that can impact the outgoing communications
- Two (three) approaches: pessimistic + reliable system, or causal, (or optimistic)
Where to save the Events?

- On a reliable media, asynchronously
- “Hope that the event will have time to be logged” (before its loss is damageable)
Where to save the Events?

- On a reliable media, asynchronously
- “Hope that the event will have time to be logged” (before its loss is damageable)
Optimistic Message Logging

Where to save the Events?
- On a reliable media, asynchronously
- “Hope that the event will have time to be logged” (before its loss is damageable)
Optimistic Message Logging

Where to save the Events?
- On a reliable media, asynchronously
- “Hope that the event will have time to be logged” (before its loss is damageable)
Optimistic Message Logging

Where to save the Events?

- On a reliable media, asynchronously
- “Hope that the event will have time to be logged” (before its loss is damageable)

herault@icl.utk.edu — yves.robert@ens-lyon.fr
Pessimistic Message Logging

Where to save the Events?

- On a reliable media, synchronously
- Delay of emissions that depend on non-deterministic choices until the corresponding choice is acknowledged
- Recovery: connect to the storage system to get the history
Pessimistic Message Logging

Where to save the Events?
- On a reliable media, synchronously
- Delay of emissions that depend on non-deterministic choices until the corresponding choice is acknowledged
- Recovery: connect to the storage system to get the history
Causal Message Logging

Where to save the Events?

- Any message carries with it (piggybacked) the whole history of non-deterministic events that precede
- Garbage collection using checkpointing, detection of cycles
- Can be coupled with asynchronous storage on reliable media to help garbage collection
- Recovery: global communication + potential storage system

Where to save the Events?

- Any message carries with it (piggybacked) the whole history of non-deterministic events that precede
- Garbage collection using checkpointing, detection of cycles
- Can be coupled with asynchronous storage on reliable media to help garbage collection
- Recovery: global communication + potential storage system

Any message carries with it (piggybacked) the whole history of non-deterministic events that precede

Garbage collection using checkpointing, detection of cycles

Can be coupled with asynchronous storage on reliable media to help garbage collection

Recovery: global communication + potential storage system
Recover in Message Logging

 Recovery

- Collect the history (from event log / event log + peers for Causal)
- Collect Id of last message sent
- Emitters resend, deliver in history order
- Fake emission of sent messages
Uncoordinated Protocol Performance

- NAS Parallel Benchmarks – 64 nodes
- High Performance Linpack
- Figures courtesy of A. Bouteiller, G. Bosilca

Overhead

First, when a non deterministic event is created, it has to be recorded for every message reception. Results are presented in Table I. Those with non-deterministic events, the performance varies significantly from those with non-deterministic events. Even on these benchmarks, the performance differences imputable to the event logging protocol only by less than 2%, which is close to the error margin for the optimistic protocol 6%

Figures courtesy of A. Bouteiller, G. Bosilca

Uncoordinated Protocol Performance
Hierarchical Protocols

Many Core Systems

- All interactions between threads considered as a message
- Explosion of number of events
- Cost of message payload logging \(\approx\) cost of communicating \(\rightarrow\) sender-based logging expensive
- Correlation of failures on the node
Hierarchical Protocols

- Processes are separated in groups
- A group co-ordinates its checkpoint
- Between groups, use message logging
Hierarchical Protocols

- Coordinated Checkpointing: the processes can behave as a non-deterministic entity (interactions between processes)
- Need to log the non-deterministic events: Hierarchical Protocols are uncoordinated protocols + event logging
- No need to log the payload
Event Log Reduction

Strategies to reduce the amount of event log

- Few HPC applications use message ordering / timing information to take decisions.
- Many receptions (in MPI) are in fact deterministic: do not need to be logged.
- For others, although the reception is non-deterministic, the order does not influence the interactions of the process with the rest (send-determinism). No need to log either.
- Reduction of the amount of log to a few applications, for a few messages: event logging can be overlapped.
Hierarchical Protocol Performance

- NAS Parallel Benchmarks – shared memory system, 32 cores
- HPL distributed system, 64 cores, 8 groups