ReViveI/O:
Efficient Handling of I/O in Highly-Available Rollback-Recovery Servers

Jun Nakano‡, Pablo Montesinos, Kourosh Gharachorloo*, Josep Torrellas

University of Illinois at Urbana-Champaign
‡IBM
*Google
Building Fault-Tolerant Systems

- Redundant Self-Checking HW
- High Overhead
- Small/Null MTTR
- HP Nonstop
- IBM S/390
Building Fault-Tolerant Systems

Motivation

Concept

Implementation

Evaluation

Conclusions

Redundant Self-Checking HW

High Overhead
Small/Null MTTR

HP Nonstop
IBM S/390

Plain HW + SW-based Checkpointing

High Overhead
Significant MTTR

KeyKOS
FT Mach
Building Fault-Tolerant Systems

Motivation

- Redundant Self-Checking HW
  - High Overhead
  - Small/Null MTTR
  - HP Nonstop
  - IBM S/390

- HW-based High-frequency Checkpointing
  - Low overhead
  - Small MTTR
  - ReVive, SafetyNet

- Plain HW + SW-based Checkpointing
  - High Overhead
  - Significant MTTR
  - KeyKOS
  - FT Mach

Concept

Implementation

Evaluation

Conclusions
<table>
<thead>
<tr>
<th>Motivation</th>
<th>Concept</th>
<th>Implementation</th>
<th>Evaluation</th>
<th>Conclusions</th>
</tr>
</thead>
</table>

A glimpse of ReVive (ISCA 2002)
A glimpse of ReVive (ISCA 2002)

- Goal: recover the memory state of a shared-memory machine in < 1s
A glimpse of ReVive (ISCA 2002)

- Goal: recover the memory state of a shared-memory machine in < 1s
- HW-assisted very frequent checkpoint (20ms < T < 100ms)
  - During checkpoints:
    - Write-back dirty data from caches
    - Main memory is the checkpoint state
  - Between checkpoints:
    - HW logs overwritten data in memory when is modified for the first time
A glimpse of ReVive (ISCA 2002)

- **Goal:** recover the memory state of a shared-memory machine in < 1s
- **HW-assisted very frequent checkpoint (20ms < T < 100ms)**
  - During checkpoints:
    - Write-back dirty data from caches
    - Main memory is the checkpoint state
  - Between checkpoints:
    - HW logs overwritten data in memory when is modified for the first time
- **Entire main memory protected by distributed parity**
  - Like RAID-5, but in memory
  - Can tolerate the loss of a node
Building Fault-Tolerant Systems

- **Motivation**
- **Concept**
- **Implementation**
- **Evaluation**
- **Conclusions**

### Case Studies

**Redundant Self-Checking HW**
- HP Nonstop
- IBM S/390
- **High Overhead**
- **Small/Null MTTR**

**HW-based High-frequency Checkpointing**
- ReVive, SafetyNet
- **Low overhead**
- **Small MTTR**

**Plain HW + SW-based Checkpointing**
- KeyKOS
- FT Mach
- **High Overhead**
- **Significant MTTR**
Building Fault-Tolerant Systems

- **Redundant Self-Checking HW**
  - Expensive

- **HW-based High-frequency Checkpointing**
  - Low overhead
  - Small MTTR
  - ReVive, SafetyNet

- **Plain HW + SW-based Checkpointing**
  - High Overhead
  - Significant MTTR
  - KeyKOS, FT Mach
Building Fault-Tolerant Systems

Motivation

Concept

Implementation

Evaluation

Conclusions

Redundant Self-Checking HW

Expensive

HW-based High-frequency Checkpointing

Low overhead Small MTTR

ReVive, SafetyNet

Plain HW + SW-based Checkpointing

Significant MTTR
Building Fault-Tolerant Systems

- **Redundant Self-Checking HW**
  - Expensive
- **HW-based High-frequency Checkpointing**
  - Modest HW
  - No I/O support
- **Plain HW + SW-based Checkpointing**
  - Significant MTTR
Building Fault-Tolerant Systems

- **Motivation**
- **Concept**
- **Implementation**
- **Evaluation**
- **Conclusions**

**Redundant Self-Checking HW**
- Expensive

**HW-based High-frequency Checkpointing**
- Modest HW
- No I/O support (until today)

**Plain HW + SW-based Checkpointing**
- Significant MTTR
Why is I/O undo/redo hard?
Why is I/O undo/redo hard?

- Output Commit Problem:
Why is I/O undo/redo hard?

• Output Commit Problem:
  • An output to the external world cannot be rolled back
Why is I/O undo/redo hard?

- Output Commit Problem:
  - An output to the external world cannot be rolled back
Why is I/O undo/redo hard?

- **Output Commit Problem:**
  - An output to the external world cannot be rolled back

![Diagram showing the difference between Checkpoint n and Checkpoint n+1 for Server, Disk, and Network.]
Why is I/O undo/redo hard?

- **Output Commit Problem:**
  - An output to the external world cannot be rolled back
Why is I/O undo/redo hard?

- **Output Commit Problem:**
  - An output to the external world cannot be rolled back
Output Commit Problem:

- An output to the external world cannot be rolled back:

```
Server       Disk
Checkpoint n  Checkpoint n+1
write        X
```

- How do you undo the I/O?
Contribution: ReViveI/O

- Support I/O undo/redo in rollback-recovery SMP servers
Contribution: ReViveI/O

- Support I/O undo/redo in rollback-recovery SMP servers
- Targeted to throughput-oriented workloads
Contribution: ReViveI/O

- Support I/O undo/redo in rollback-recovery SMP servers
- Targeted to throughput-oriented workloads
  - Disk and network only
Contribution: ReViveI/O

- Support I/O undo/redo in rollback-recovery SMP servers
- Targeted to throughput-oriented workloads
  - Disk and network only
- Transparent to OS and applications
Contribution: ReViveI/O

- Support I/O undo/redo in rollback-recovery SMP servers
- Targeted to throughput-oriented workloads
  - Disk and network only
- Transparent to OS and applications
- Low overhead during fault-free operation (<1% over ReVive)
Contribution: ReViveI/O

- Support I/O undo/redo in rollback-recovery SMP servers
- Targeted to throughput-oriented workloads
  - Disk and network only
- Transparent to OS and applications
- Low overhead during fault-free operation (<1% over ReVive)
- Fast recovery (< 1s even if processor lost)
Contribution: ReVive I/O

- Support I/O undo/redo in rollback-recovery SMP servers
- Targeted to throughput-oriented workloads
  - Disk and network only
- Transparent to OS and applications
- Low overhead during fault-free operation (<1% over ReVive)
- Fast recovery (< 1s even if processor lost)
- Small impact on hardware cost
Approach: Pseudo-Device Driver

Masubuchi et al, FTCS-97
Approach: Pseudo-Device Driver

Masubuchi et al, FTCS-97
Approach: Pseudo-Device Driver

Masubuchi et al, FTCS-97
Approach: Pseudo-Device Driver

Masubuchi et al, FTCS-97
Approach: Pseudo-Device Driver

- Kernel
- Pseudo Device-Driver (PDD)
- Device Driver
- Device

Masubuchi et al, FTCS-97
Approach: Pseudo-Device Driver

- Delays outputs until next checkpoint
- Commits outputs after next checkpoint
- No need to modify OS nor applications
- Network and Disk PDDs

Masubuchi et al, FTCS-97
Fault Model
Fault Model

Fault!
Fault Model

MR = Transients or 1 permanent with at most 1 node loss
Fault Model

Fault!

Memory Recoverable by ReVive (MR)

Yes

ReVive restores memory from previous checkpoint

PDD recovers I/O transparently

MR = Transients or 1 permanent with at most 1 node loss
Fault Model

MR = Transients or 1 permanent with at most 1 node loss
Fault Model

- **Fault!**
  - **Yes**: Memory Recoverable by ReVive (MR)
    - ReVive restores memory from previous checkpoint
    - PDD recovers I/O transparently
  - **No**: Fix HW and Reboot

MR=Transients or 1 permanent with at most 1 node loss
Fault Model

Fault!

Memory Recoverable by ReVive (MR)

Yes: ReVive restores memory from previous checkpoint
PDD recovers I/O transparently

No: Fix HW and Reboot
PDD makes the I/O state consistent

MR=Transients or 1 permanent with at most 1 node loss
Fault Model

Fault!

Yes

Memory Recoverable by ReVive (MR)

ReVive restores memory from previous checkpoint

PDD recovers I/O transparently

No

Fix HW and Reboot

PDD makes the I/O state consistent

Conventional recovery by Application (e.g. DB)

MR=Transients or 1 permanent with at most 1 node loss
Network PDD and TCP

Motivation

Concept

Implementation

Evaluation

Conclusions
Network PDD and TCP

Motivation

Concept

Implementation

Evaluation

Conclusions

Network PDD and TCP:

- Network PDD
- Kernel, App
- Client

Time

CKPT_i, CKPT_{i+1}, CKPT_{i+2}, CKPT_{i+3}, CKPT_{i+4}, CKPT_{i+5}
- ReViveI/O leverages the properties of TCP to support network undo/redo
ReVive I/O leverages the properties of TCP to support network undo/redo
ReViveI/O leverages the properties of TCP to support network undo/redos.
ReViveI/O leverages the properties of TCP to support network undo/redos.
ReViveI/O leverages the properties of TCP to support network undo/redo.
Network PDD and TCP

- ReViveI/O leverages the properties of TCP to support network undo/redo
ReViveI/O leverages the properties of TCP to support network undo/redo
ReViveI/O leverages the properties of TCP to support network undo/redo.

TCP avoids saving packages for inputs: the sender timeouts and retransmits.
- ReViveI/O leverages the properties of TCP to support network undo/redo
- TCP avoids saving packages for inputs: the sender timeouts and retransmits
Network PDD and TCP

- ReViveI/O leverages the properties of TCP to support network undo/redos.
- TCP avoids saving packages for inputs: the sender timeouts and retransmits.
- ReViveI/O leverages the properties of TCP to support network undo/redo
- TCP avoids saving packages for inputs: the sender timeouts and retransmits
ReViveI/O leverages the properties of TCP to support network undo/redo
- TCP avoids saving packages for inputs: the sender timeouts and retransmits
ReViveI/O leverages the properties of TCP to support network undo/redo.

TCP avoids saving packages for inputs: the sender timeouts and retransmits.
ReViveI/O leverages the properties of TCP to support network undo/redo
- TCP avoids saving packages for inputs: the sender timeouts and retransmits
ReViveI/O leverages the properties of TCP to support network undo/redo

TCP avoids saving packages for inputs: the sender timeouts and retransmits
• ReViveI/O leverages the properties of TCP to support network undo/redo
• TCP avoids saving packages for inputs: the sender timeouts and retransmits
ReViveI/O leverages the properties of TCP to support network undo/redo

TCP avoids saving packages for inputs: the sender timeouts and retransmits
Network PDD and TCP

- ReViveI/O leverages the properties of TCP to support network undo/redo
- TCP avoids saving packages for inputs: the sender timeouts and retransmits
ReVive I/O leverages the properties of TCP to support network undo/redo:

- TCP avoids saving packages for inputs: the sender timeouts and retransmits.
- TCP eliminates duplicate packages.
ReViveI/O leverages the properties of TCP to support network undo/redo

- TCP avoids saving packages for inputs: the sender timeouts and retransmits
- TCP eliminates duplicate packages
### Disk PDD Schemes

<table>
<thead>
<tr>
<th>Scheme</th>
<th>On I/O write issue</th>
<th>After next checkpoint</th>
</tr>
</thead>
</table>

## Disk PDD Schemes

<table>
<thead>
<tr>
<th>Scheme</th>
<th>On I/O write issue</th>
<th>After next checkpoint</th>
</tr>
</thead>
<tbody>
<tr>
<td>Stall</td>
<td>Stall the write</td>
<td>Write data to disk</td>
</tr>
<tr>
<td>Scheme</td>
<td>On I/O write issue</td>
<td>After next checkpoint</td>
</tr>
<tr>
<td>--------</td>
<td>--------------------</td>
<td>-----------------------</td>
</tr>
<tr>
<td>Stall</td>
<td>Stall the write</td>
<td>Write data to disk</td>
</tr>
</tbody>
</table>
## Disk PDD Schemes

<table>
<thead>
<tr>
<th>Scheme</th>
<th>On I/O write issue</th>
<th>After next checkpoint</th>
</tr>
</thead>
<tbody>
<tr>
<td>Stall</td>
<td>Stall the write</td>
<td>Write data to disk</td>
</tr>
<tr>
<td>Logging</td>
<td>Copy old data elsewhere in disk</td>
<td>Delete pointer to old data</td>
</tr>
<tr>
<td></td>
<td>Write data in place</td>
<td></td>
</tr>
</tbody>
</table>
## Disk PDD Schemes

<table>
<thead>
<tr>
<th>Scheme</th>
<th>On I/O write issue</th>
<th>After next checkpoint</th>
</tr>
</thead>
<tbody>
<tr>
<td>Stall</td>
<td>Stall the write</td>
<td>Write data to disk</td>
</tr>
<tr>
<td>Logging</td>
<td>Copy old data elsewhere in disk</td>
<td>Delete pointer to old data</td>
</tr>
<tr>
<td></td>
<td>Write data in place</td>
<td></td>
</tr>
<tr>
<td>Buffering</td>
<td>Write data to disk buffer and mem</td>
<td>Copy data from mem to disk</td>
</tr>
</tbody>
</table>
## Disk PDD Schemes

<table>
<thead>
<tr>
<th>Scheme</th>
<th>On I/O write issue</th>
<th>After next checkpoint</th>
</tr>
</thead>
<tbody>
<tr>
<td>Stall</td>
<td>Stall the write</td>
<td>Write data to disk</td>
</tr>
<tr>
<td>Logging</td>
<td>Copy old data elsewhere in disk</td>
<td>Delete pointer to old data</td>
</tr>
<tr>
<td></td>
<td>Write data in place</td>
<td></td>
</tr>
<tr>
<td>Buffering</td>
<td>Write data to disk buffer and mem</td>
<td>Copy data from mem to disk</td>
</tr>
<tr>
<td>Renaming</td>
<td>Write data to renamed disk location</td>
<td>Delete old mapping</td>
</tr>
<tr>
<td></td>
<td>Save old logical → physical mapping</td>
<td></td>
</tr>
</tbody>
</table>
## Disk PDD Schemes

<table>
<thead>
<tr>
<th>Scheme</th>
<th>On I/O write issue</th>
<th>After next checkpoint</th>
</tr>
</thead>
<tbody>
<tr>
<td>Stall</td>
<td>Stall the write</td>
<td>Write data to disk</td>
</tr>
<tr>
<td>Logging</td>
<td>Copy old data elsewhere in disk</td>
<td>Delete pointer to old data</td>
</tr>
<tr>
<td>Buffering</td>
<td>Write data to disk buffer and mem</td>
<td>Copy data from mem to disk</td>
</tr>
<tr>
<td>Renaming</td>
<td>Write data to renamed disk location</td>
<td>Delete old mapping</td>
</tr>
<tr>
<td></td>
<td>Save old logical → physical mapping</td>
<td></td>
</tr>
</tbody>
</table>
Disk PDD

Client

update request

Network PDD

time

Kernel, App

Disk PDD

CKPT_i

Disk Buffer

Memory Buffer

Target Disk

CKPT_{i+2}

CKPT_{i+3}

CKPT_{i+4}

CKPT_{i+5}
Disk PDD

Client

Network PDD

Kernel, App

Disk PDD

Time

update request

CKPT\textsubscript{i}

CKPT\textsubscript{i+2}

CKPT\textsubscript{i+3}

CKPT\textsubscript{i+4}

CKPT\textsubscript{i+5}

Disk Buffer

Memory Buffer

Target Disk

Network PDD
Disk PDD

Client

update request

Network PDD

Kernel, App

Disk PDD

CKPT\_i

CKPT\_i+2

CKPT\_i+3

CKPT\_i+4

CKPT\_i+5

Disk Buffer

Memory Buffer

Target Disk
Disk PDD

- **Client**: update request
- **Network PDD**: CKPT\(_i\) to CKPT\(_i+5\)
- **Kernel, App**: CKPT\(_i+2\) to CKPT\(_i+5\)
- **Disk PDD**: CKPT\(_i+3\) to CKPT\(_i+5\)

**MR Fault**
- Memory Buffer
- Disk Buffer
- Target Disk

**Time**
Disk PDD

Disk PDD

Network PDD

Kernel, App

Disk PDD

Client

MR Fault

Disk Buffer

Memory Buffer

Target Disk

CKPT

CKPT

CKPT

CKPT

CKPT

Time

CKPT

CKPT

CKPT

CKPT
Disk PDD

Client

Network PDD
Kernel, App
Disk PDD

CKPT\textsubscript{i}
CKPT\textsubscript{i+2}
CKPT\textsubscript{i+3}
CKPT\textsubscript{i+4}
CKPT\textsubscript{i+5}

update request

MR Fault

Disk Buffer
Memory Buffer
Target Disk

Time
Disk PDD

- Motivation
- Concept
- Implementation
- Evaluation
- Conclusions

Client

Network PDD

Kernel, App

Disk PDD

CKPT\textsubscript{i}

CKPT\textsubscript{i+2}

CKPT\textsubscript{i+3}

CKPT\textsubscript{i+4}

CKPT\textsubscript{i+5}

Time

update request

Disk Buffer

Memory Buffer

Target Disk

MR Fault

Disk PDD

Network PDD

Client

Memory Buffer

Disk Buffer

Target Disk

Update request

Disk PDD

- Client
- Network PDD
- Kernel, App
- Disk PDD

Update request:
- CKPT\textsubscript{i}
- CKPT\textsubscript{i+2}
- CKPT\textsubscript{i+3}
- CKPT\textsubscript{i+4}
- CKPT\textsubscript{i+5}

- Memory Buffer
- Disk Buffer
- Target Disk

MR Fault
Disk PDD

Motivation
Concept
Implementation
Evaluation
Conclusions

Client

Network PDD
Kernel, App
Disk PDD

update request

CKPT_i
CKPT_{i+2}
CKPT_{i+3}
CKPT_{i+4}
CKPT_{i+5}

Memory Buffer
Disk Buffer
Target Disk

MR Fault
Disk PDD

Client

Network PDD

Kernel, App

Disk PDD

update request

CKPT_i

CKPT_{i+2}

CKPT_{i+3}

CKPT_{i+4}

CKPT_{i+5}

Memory Buffer

Disk Buffer

Target Disk

MR Fault
Disk PDD
Disk PDD

Client

Network PDD

Kernel, App

Disk PDD

CKPT_{i}

CKPT_{i+2}

CKPT_{i+3}

CKPT_{i+4}

CKPT_{i+5}

update request

response

MR Fault

Disk Buffer

Memory Buffer

Target Disk

Time
Disk PDD

Motivation

Concept

Implementation

Evaluation

Conclusions

Disk PDD

Client

Network PDD

Kernel, App

Disk PDD

CKPT\textsubscript{i}

CKPT\textsubscript{i+2}

CKPT\textsubscript{i+3}

CKPT\textsubscript{i+4}

CKPT\textsubscript{i+5}

Time

update request

Disk Buffer

Memory Buffer

Target Disk

MR Fault

Disk PDD

Network PDD

Target Disk

CKPT

Client

update request
Disk PDD

Motivation

Concept

Implementation

Evaluation

Conclusions

Client

Network PDD

Kernel, App

Disk PDD

CKPT\textsubscript{i}

CKPT\textsubscript{i+2}

CKPT\textsubscript{i+3}

CKPT\textsubscript{i+4}

CKPT\textsubscript{i+5}

update request

response

Disk Buffer

Memory Buffer

Target Disk

MR Fault

update request

response

Disk Buffer

Memory Buffer

Target Disk

Disk PDD

- **Client**
  - **CKPT\textsubscript{i}
  - **CKPT\textsubscript{i+2}
  - **CKPT\textsubscript{i+3}
  - **CKPT\textsubscript{i+4}
  - **CKPT\textsubscript{i+5}

- **Network PDD**
- **Kernel, App**
- **Disk PDD**

- **Disk Buffer**
- **Memory Buffer**
- **Target Disk**

**Time**

- **update request**
- **response**

**Markers**:
- **MR Fault**
- **Non MR Fault**

**Diagram Notes**:
- Disk PDD
- Network PDD
- Kernel, App
- Client
- Time
- Update request
- Response
- MR Fault
- Non MR Fault
Disk PDD

Client

Network PDD

Kernel, App

Disk PDD

CKPT\(_i\)

CKPT\(_{i+2}\)

CKPT\(_{i+3}\)

CKPT\(_{i+4}\)

CKPT\(_{i+5}\)

update request

---

Memory Buffer

Disk Buffer

Target Disk

MR Fault

Non MR Fault

Time

Disk PDD

- **Network PDD**
- **Kernel, App**
- **Disk PDD**

- **Client**

- **Time**

- **CKPT_{i}**, **CKPT_{i+2}**, **CKPT_{i+3}**, **CKPT_{i+4}**, **CKPT_{i+5}**

- **Disk Buffer**
- **Target Disk**

- **Yellow** MR Fault
- **Red** Non MR Fault

---

Disk PDD

Client

Network PDD

Kernel, App

Disk PDD

CKPT\textsubscript{i}

CKPT\textsubscript{i+2}

CKPT\textsubscript{i+3}

CKPT\textsubscript{i+4}

CKPT\textsubscript{i+5}

Disk Buffer

Target Disk

\textcolor{yellow}{MR Fault}

\textcolor{red}{Non MR Fault}

Recovery: ReVive + ReViveI/O

Motivation  Concept  Implementation  Evaluation  Conclusions
Recovery: ReVive + ReViveI/O

*Useful Work*

- CKP

*Time*
Recovery: ReVive + ReViveI/O

Useful Work
Recovery: ReVive + ReViveI/O

Lost Work

Useful Work

Detection

CKP

Time

100ms

80ms
Recovery: ReVive + ReViveI/O

Time

100ms

80ms

50ms

CKP

Detection

• Self-Check

• Rerouting

Useful Work

Lost Work
Recovery: ReVive + ReViveI/O

**Time**
- 100ms
- 80ms
- 50ms
- ~100ms

**CKP**
- Detection
- Self-Check
- Rerouting
- Reconstruct Lost Log

**Lost Work**

**Useful Work**

**ReVive Recovery**
Recovery: ReVive + ReViveI/O

**Time**
- 100ms
- 80ms
- 50ms
- ~100ms
- ~490ms

**CKP**
- Detection
- Self-Check
- Rerouting
- Reconstruct Lost Log
- Rollback

**Useful Work**
**Lost Work**
**ReVive Recovery**
Recovery: ReVive + ReViveI/O

**Motivation**

**Concept**

**Implementation**

**Evaluation**

**Conclusions**

---

**Useful Work**

**Lost Work**

**ReVive Recovery**
Recovery: ReVive + ReViveI/O

**Time**
- Machine Unavailable:
  - 100ms (Detection)
  - 80ms (Self-Check, Rerouting)
  - 50ms (Reconstruct Lost Log)
  - ~100ms (Rollback)
  - ~490ms (Reconstruct Lost Data)
- Degraded Execution:
  - ~seconds

**Useful Work**
- Useful Work

**Lost Work**
- Lost Work

**ReVive Recovery**
- ReVive I/O Recovery
Recovery: ReVive + ReViveI/O

- **Machine Unavailable**
  - 100ms
  - Detection
  - Self-Check
  - Rerouting
  - Reconstruct Lost Log: ~100ms
  - Rollback: ~490ms

- **Degraded Execution**
  - ~seconds
  - Reconstruct Lost Data

- **Useful Work**
  - ~100ms
  - CKP

- **Lost Work**
  - ~100ms
  - Detection
  - Self-Check
  - Rerouting

- **ReVive Recovery**
  - Reset Device (~1ms)

- **ReVive I/O Recovery**
Recovery: ReVive + ReViveI/O

- **Lost Work**
- **Useful Work**
- **ReVive Recovery**

**Machine Unavailable**
- 100ms
- 80ms
- 50ms
- ~100ms
- ~490ms
- ~seconds

**Detection**
- 100ms
- 80ms
- 50ms
- ~100ms
- ~490ms
- ~seconds

- **Self-Check**
- **Rerouting**

**Reconstruct**
- Lost Log
- ~100ms
- ~490ms
- ~seconds

**Rollback**
- ~100ms
- ~490ms
- ~seconds

**Reconstruct Lost Data**
- ~seconds

**Reset Device**
- (~1ms)

**Reset Device Driver**
- (~1ms)

**ReVive I/O Recovery**

- Useful Work
- Lost Work
- ReVive Recovery
Recovery: ReVive + ReViveI/O

Motivation
Concept
Implementation
Evaluation
Conclusions

ReVive Recovery

Useful Work
Lost Work
ReVive Recovery

Detection • Self-Check • Rerouting
Reconstruct Lost Log
Rollback
Reconstruct Lost Data
Re-issuing Outputs in the Background (<100ms)

Reset Device
Reset Device Driver

ReVive I/O Recovery

Machine Unavailable
Degraded Execution

Time

100ms
80ms
50ms
~100ms
~490ms
~seconds

Lost Work

CKP

100ms
80ms
50ms
~100ms
~490ms
~seconds
Experiments

Clients

• Dual 1.5 GHz Athlon
• Gigabit Ethernet
• 1 GB RAM
Experiments

- Dual 1.5 GHz Athlon
- Gigabit Ethernet
- 1 GB RAM

Clients

Server

- Dual 1.5 GHz Athlon
- Gigabit Ethernet
- 2 GB RAM

+ ReVive PDD
Throughput-oriented workload

TPC-C on Oracle9i with 30 Clients

Checkpoint interval (ms)
Throughput-oriented workload

TPC-C on Oracle9i with 30 Clients

Checkpoint interval (ms)

Normalized Throughput (%)

- 1.1
- 1.0
- 0.9
- 0.8
- 0.7
- 0.6
- 0.5
- 0.4
- 0.3
- 0.2
- 0.1
- 0

TPC-C on Oracle9i with 30 Clients
Throughput-oriented workload

TPC-C on Oracle9i with 30 Clients

- Stall
- ReVive/O

Checkpoint interval (ms) vs Normalized Throughput (%)
Throughput-oriented workload

- Stalling scheme (simply delaying writes) incurs too much overhead
Throughput-oriented workload

- Stalling scheme (simply delaying writes) incurs too much overhead
- ReViveI/O’s overhead is only 0.1% for T<120 ms
Integration with ReVive (for TPC-C)

- Checkpoint Interval (ms)
- Reduction in Throughput (%)
Integration with ReVive (for TPC-C)

- **Motivation**
- **Concept**
- **Implementation**
- **Evaluation**
- **Conclusions**

![Graph showing reduction in throughput vs checkpoint interval (ms)]
Integration with ReVive (for TPC-C)

![Graph showing the reduction in throughput (%)]

- **ReVive + ReVive I/O**
- **ReVive Only (estimated)**

- **Axes:**
  - Y-axis: Reduction in Throughput (%)
  - X-axis: Checkpoint Interval (ms)

The graph illustrates the reduction in throughput (%) as a function of the checkpoint interval (ms) for two scenarios:

- **ReVive + ReVive I/O**
- **ReVive Only (estimated)**
Integration with ReVive (for TPC-C)

- ReVive + ReVive I/O: only 7% throughput reduction for 60 - 120 ms checkpoint intervals
Availability

<table>
<thead>
<tr>
<th>MTBF</th>
<th>1 hr</th>
<th>2 hr</th>
<th>4 hr</th>
<th>1 day</th>
<th>1 week</th>
<th>1 month</th>
<th>3 months</th>
<th>6 months</th>
<th>1 year</th>
<th>10 years</th>
</tr>
</thead>
<tbody>
<tr>
<td>99.9%</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>99.99%</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>99.999%</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>99.9999%</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>99.99999%</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Availability

Baseline (No Rollback)

MTBF<sub>MR</sub>

Availability

- 99.9%
- 99.99%
- 99.999%
- 99.9999%
- 99.99999%

- 1 hr
- 2
- 4 hr
- 1 day
- 1 week
- 1 month
- 3
- 6
- 1 year
- 10 years

Availability

Baseline (No Rollback)

ReVive + ReViveI/O. MTBF_{NMR} = 1000 \times MTBF_{MR}

MTBF_{MR}

Availability

1 hr 2 4 hr 1 day 1 week 1 month 3 6 1 year 10 years

99.9%
99.99%
99.999%
99.9999%
99.99999%
99.999999%

MTBF_{MR}
ReVive I/O beats baseline thanks to its low recovery latency for MR faults.
Conclusions

- Support for I/O undo/redo in memory-checkpointing systems:
  - Transparent to OS and applications
  - Fast recovery and negligible overhead during fault-free operation
  - Low space overhead (see paper)
Support for I/O undo/redo in memory-checkpointing systems:

- Transparent to OS and applications
- Fast recovery and negligible overhead during fault-free operation
- Low space overhead (see paper)

Troughtput-oriented workloads:

- < 1% overhead over ReVive
Conclusions

• Support for I/O undo/redo in memory-checkpointing systems:
  • Transparent to OS and applications
  • Fast recovery and negligible overhead during fault-free operation
  • Low space overhead (see paper)

• Throughput-oriented workloads:
  • < 1% overhead over ReVive

• Latency-bound workloads:
  • Tolerable checkpoint interval is application dependent (see paper)
Conclusions

- Support for I/O undo/redo in memory-checkpointing systems:
  - Transparent to OS and applications
  - Fast recovery and negligible overhead during fault-free operation
  - Low space overhead (see paper)

- Throughput-oriented workloads:
  - < 1% overhead over ReVive

- Latency-bound workloads:
  - Tolerable checkpoint interval is application dependent (see paper)

- ReVive I/O can be used for transactional memory
ReVive I/O:
Efficient Handling of I/O in Highly-Available Rollback-Recovery Servers

Jun Nakano‡, Pablo Montesinos, Kourosh Gharachorloo*, Josep Torrellas

University of Illinois at Urbana-Champaign
‡IBM
*Google
Distributed N+1 Parity in ReVive

Motivation  Concept  Implementation  Evaluation  Conclusions
Distributed N+1 Parity in ReVive

Motivation

Concept

Implementation

Evaluation

Conclusions
Distributed N+1 Parity in ReVive

Parity Group
  Parity
  Data
  Data

Node O
Node 1
Node N

...
Distributed N+1 Parity in ReVive

- Allocation granularity: page
- Update granularity: cache line
- Distributed parity: can recover from loss of one node
Space Overhead

- **Network PDD:**
  - Need to increase OS’s TCP sliding window size
  - e.g., Gigabit Ethernet, $T = 100\,\text{ms} \rightarrow 12\,\text{MB}$

- **Disk PDD (buffering):**
  - Disk PDD needs private memory area for buffering
  - e.g., 40 MB/s disk, $T = 100\,\text{ms} \rightarrow 4\,\text{MB}$
Latency-oriented workload
Latency-oriented workload

WebStone + Apache

Response Time (ms)

Number of Clients

Latency-oriented workload

WebStone + Apache

Number of Clients

Response Time (ms)

- Baseline
- Buffering - 20ms CKPT
- Buffering - 40ms CKPT
- Buffering - 80ms CKPT
- Buffering - 160ms CKPT
Latency-oriented workload

- Tolerable checkpoint interval is application dependent
- Response times up to 100ms are acceptable for interactive applications
## Disk PDD Schemes

<table>
<thead>
<tr>
<th></th>
<th>Stall</th>
<th>Concept</th>
<th>Implementation</th>
<th>Evaluation</th>
<th>Conclusions</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>On I/O write issue</td>
<td>Stalls the write</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>After next CKPT</td>
<td>Writes data to disk</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>On I/O write issue</td>
<td>Copies old data elsewhere in disk</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Writes data in place</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>After next CKPT</td>
<td>Deletes pointer to old data</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Logging</td>
<td>On I/O write issue</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>After next CKPT</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Buffering</td>
<td>On I/O write issue</td>
<td>Writes data to disk buffer and mem</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>After next CKPT</td>
<td>Copies data from mem to disk</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Renaming</td>
<td>On I/O write issue</td>
<td>Writes data to renamed disk location</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Saves old logical → physical mapping</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>After next CKPT</td>
<td>Delete old mapping</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Network PDD and TCP

- Resend packets after rollback
- TCP eliminates duplicates

- Can avoid saving inputs replay
- Sender timeouts and resend
Network PDD and TCP

- Resend packets after rollback
- TCP eliminates duplicates

- Can avoid saving inputs replay
- Sender timeouts and resend
Network PDD and TCP

- Resend packets after rollback
- TCP eliminates duplicates
- Can avoid saving inputs replay
- Sender timeouts and resend
Network PDD and TCP

- Resend packets after rollback
- TCP eliminates duplicates
- Can avoid saving inputs replay
- Sender timeouts and resend
Network PDD and TCP

- Resend packets after rollback
- TCP eliminates duplicates
- Can avoid saving inputs replay
- Sender timeouts and resend
Network PDD and TCP

- Resend packets after rollback
- TCP eliminates duplicates
- Can avoid saving inputs replay
- Sender timeouts and resend
Resend packets after rollback
TCP eliminates duplicates
Can avoid saving inputs replay
Sender timeouts and resend
Network PDD and TCP

- Resend packets after rollback
- TCP eliminates duplicates
- Can avoid saving inputs replay
- Sender timeouts and resend
Integration with ReVive

- Simple model of ReVive overhead:
  - $c$ and $r$ are constants (regardless of checkpoint frequency)
  - For instance, $c = 1$ ms, $r=5\%$ and $T = 100$ ms give 6% overhead
Integration with ReVive (WebStone)

- Increase in a latency-bound workload is $2 \times T$ (Checkpoint Interval)
- Most of the overhead comes from ReVive I/O