Tardigrade: Leveraging Lightweight Virtual Machines to Easily and Efficiently Construct Fault-Tolerant Services

Jacob R. Lorch   Andrew Baumann

Microsoft Research

Lisa Glendenning   Dutch T. Meyer   Andrew Warfield
Our goal: 

*Turn existing* binaries into fault-tolerant services.
Example: FDS Metadata Service

[Example Image of FDS Cluster]

FDS Cluster

FDS Metadata server

[Nightingale et al., OSDI 2012]
Example: FDS Metadata Service

FDS Cluster

FDS Metadata server

Paxos leader election

not safe because two active metadata servers will cause cluster corruption. Our current solution is simple: only one metadata server is allowed to execute at a time. If it fails, an operator must ensure the old one is decommissioned and start a new one. We are experimenting with using Paxos to safely make this process automatic and reduce the impact to availability. Once a

[Nightingale et al., OSDI 2012]
Techniques for making code fault-tolerant

Use state machine replication library

Better: *Transparency makes the binary fault-tolerant*

Potential for oversight
- Non-determinism
- Failing to persist state
- Exposing non-persisted data
- Bugs in crash recovery
Outline

• Motivation
• Background: Asynchronous VM replication
• Our solution: Lightweight VM replication
• Challenges and solutions
• Evaluation
Outline

• Motivation
• **Background**: Asynchronous VM replication
• Our solution: Lightweight VM replication
• Challenges and solutions
• Evaluation
Asynchronous virtual machine replication - Remus

Primary can crash at any time; backup is always a bit behind.

[Cully et al., NSDI 2008]
Asynchronous virtual machine replication - Remus

[Cully et al., NSDI 2008]
Asynchronous virtual machine replication - Remus

[Cully et al., NSDI 2008]
High VM activity can delay packets

Processes unrelated to the service can balloon client-perceived latency.
Outline

• Motivation
• Background: Asynchronous VM replication
• Our solution: Lightweight VM replication
• Challenges and solutions
• Evaluation
Our solution: Use *lightweight* VMs instead

**Lightweight VM system examples**
- Xax [Douceur et al., OSDI 2008]
- Native Client [Sehr et al., IEEE S&P 2009]
- Drawbridge [Porter et al., ASPLOS 2011]
- Embassies [Howell et al., NSDI 2013]
- Bascule [Baumann et al., Eurosys 2013]

Narrow API (e.g., ~45 calls in Bascule)
Lightweight VMs can support unmodified binaries via a library OS
Lightweight VMs can support unmodified binaries via a library OS

Bascule has a Windows LibOS and a Linux LibOS
A lightweight VM is encapsulated by virtue of having a narrow interface
Our approach: Checkpoint by interposing on existing LVM API

Interposition using existing API means LVM and LibOS don’t have to change
Asynchronous Virtual Machine Replication

Lightweight Virtual Machine Replication

[Cully et al., NSDI 2008]
Asynchronous Virtual Machine Replication

Lightweight Virtual Machine Replication

Our implementation of LVMR is called Tardigrade

primary

backup

Checkpointing Host

Checkpointing Host

primary

backup
Outline

• Motivation
• Background: Asynchronous VM replication
• Our solution: Lightweight VM replication
• Challenges and solutions
• Evaluation
Practical LVMR poses challenges

Challenges

- Maintaining consistency across reconfigurations
- Achieving performance potential
- Checkpointing via an existing LVM API

Solutions

- Vertical Paxos
- Incremental checkpointing, pre-checkpointing, parallelism, scaling send buffer size
- Quiescing, pre-checkpointing, enforcing determinism, terminating connections

Lessons for LVM API designers

See paper for details
### Checkpointing uses certain LVM API features

<table>
<thead>
<tr>
<th>Feature</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ability to track changed memory pages</td>
<td>Efficiently compute checkpoint deltas</td>
</tr>
<tr>
<td>Ability to suspend and inspect other threads</td>
<td>Capture consistent snapshot</td>
</tr>
<tr>
<td>Determinism when API calls are replayed</td>
<td>Prevent divergence on failover</td>
</tr>
<tr>
<td>Host state either replayable or regeneratable</td>
<td>Recreate host state on backup</td>
</tr>
</tbody>
</table>
Features may not always be in LVM APIs

<table>
<thead>
<tr>
<th>Feature</th>
<th>Workaround</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ability to track changed memory pages</td>
<td>Use exceptions, pre-checkpointing</td>
</tr>
<tr>
<td><strong>Missing</strong> ability to suspend and inspect other threads</td>
<td>Hide non-determinism</td>
</tr>
<tr>
<td><strong>Non-determinism</strong> when API calls are replayed</td>
<td>Expose divergence as error condition</td>
</tr>
<tr>
<td>Host state <strong>not</strong> replayable or regeneratable</td>
<td></td>
</tr>
</tbody>
</table>
To capture a checkpoint, we must quiesce and capture all threads’ state.

What if the API doesn’t let a thread suspend and inspect another thread?
We can use exceptions to quiesce guest threads.
Exception handler quiesces and captures each guest thread’s state

ExceptionHandler(, )

Checkpointing layer

Guest (service + library OS)

Host

Memory

Checkpoint

primary
Synchronous system calls complicate quiescence

- Guest (service + library OS)
- Checkpointing layer
- Host

Primary
The wait system call is easy to deal with

- **Guest (service + library OS)**
- **Checkpointing layer**
- **Host**

<table>
<thead>
<tr>
<th>select() file descriptor list</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x1AC</td>
</tr>
<tr>
<td>0x3BB</td>
</tr>
<tr>
<td>0x907</td>
</tr>
<tr>
<td>time-to-checkpoint</td>
</tr>
</tbody>
</table>
General synchronous system calls require *pre-checkpointing*
API non-determinism undermines replay

CreateSemaphore() returns descriptor 0xAAA

CreateSemaphore() returns descriptor 0xBBB
An indirection table can hide non-determinism

Guest (service + library OS)

Checkpointing layer

<table>
<thead>
<tr>
<th>Guest descriptor</th>
<th>Host descriptor</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x002</td>
<td>0x932</td>
</tr>
</tbody>
</table>

Host

primary

Guest (service + library OS)

Checkpointing layer

<table>
<thead>
<tr>
<th>Guest descriptor</th>
<th>Host descriptor</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x002</td>
<td>0x909</td>
</tr>
</tbody>
</table>

Host

backup

Jay Lorch, Microsoft Research
State external to guest needs to be replayable or regeneratable

Guest (service + library OS)

Checkpointing layer

Host

TCP session state

API provides sockets, not packets

Checkpointer can’t capture TCP session state!

LVM API
System-specific modifications may be necessary

TCP connections get dropped on a failover.

Fixing this requires a major API change to make it use packets rather than sockets.
Outline

• Motivation
• Background: Asynchronous VM replication
• Our solution: Lightweight VM replication
• Challenges and solutions
• Evaluation
Effect of external processes - Remus

Latency of ping (ms)

- 50th quantile
- 95th quantile
- 99th quantile
- 99.9th quantile

Baseline  | Safety Scan  | Search Indexer  | Update  | Deduplication
---|---|---|---|---
43  | 66  | 67  | 77  | 81.9
71  | 96  | 102 | 722 | 4942
76  | 104 | 160 | 1716 | 7741
88  | 151 | 276 | 2460 | 9697

Jay Lorch, Microsoft Research

Tardigrade
Effect of external processes - Tardigrade

Latency (ms)

Quantile

50% 95% 99% 99.9%

Baseline Safety Scan Search Indexer Update Deduplication
Effect of external processes - Tardigrade

![Bar chart showing latency (ms) for different quantiles and processes]

- Baseline
- Safety Scan
- Search Indexer
- Update
- Deduplication

Jay Lorch, Microsoft Research
Memory dirtying affects checkpoint latency

No dirtying  10% of net b/w  20% of net b/w
30% of net b/w  40% of net b/w  50% of net b/w
FDS metadata service

- Metadata server initially idle
- Cluster starting up
- Cluster operating normally

Checkpoint delta average size: 0.9 MB

Checkpoint delta average size: 1.8 MB
ZKLite, a simple non-fault-tolerant Java implementation of the Zookeeper API
Conclusions

No changes to binaries needed, making deployment simple

Reasonable performance if memory dirtying rate and load are low

Lightweight VM replication is practical for making existing service binaries fault-tolerant