PLOVER: Fast, Multi-core Scalable Virtual Machine Fault-tolerance

Cheng Wang, Xusheng Chen, Weiwei Jia, Boxuan Li, Haoran Qiu, Shixiong Zhao, and Heming Cui
The University of Hong Kong
Virtual machines are pervasive in datacenters

VM fault tolerance is crucial!
Classic VM replication - primary/backup approach

Remus [NSDI’08]
Classic VM replication - primary/backup approach

Remus [NSDI’08]
Classic VM replication - primary/backup approach

Remus [NSDI’08]
Synchronize primary/backup every 25ms
1. Pause primary VM (every 25ms) and transmit all changed state (e.g., memory pages) to backup.
Synchronize primary/backup every 25ms
1. Pause primary VM (every 25ms) and transmit all changed state (e.g., memory pages) to backup.
Classic VM replication - primary/backup approach

Synchronize primary/backup every 25ms
1. Pause primary VM (every 25ms) and transmit all changed state (e.g., memory pages) to backup.
Synchronize primary/backup every 25ms
1. Pause primary VM (every 25ms) and transmit all changed state (e.g., memory pages) to backup.
Synchronize primary/backup every 25ms
1. Pause primary VM (every 25ms) and transmit all changed state (e.g., memory pages) to backup.
Synchronize primary/backup every 25ms

1. Pause primary VM (every 25ms) and transmit all changed state (e.g., memory pages) to backup.
Classic VM replication - primary/backup approach

Synchronize primary/backup every 25ms
1. Pause primary VM (every 25ms) and transmit all changed state (e.g., memory pages) to backup.
2. Backup acknowledges to the primary when complete state has been received.
Classic VM replication - primary/backup approach

Remus [NSDI’08]

Primary

Guest VM

memory pages

service

ACK

VMM

Output buffer

backup

Guest VM

memory pages

service

VMM

client

Synchronize primary/backup every 25ms

1. Pause primary VM (every 25ms) and transmit all changed state (e.g., memory pages) to backup.
2. Backup acknowledges to the primary when complete state has been received.
Classic VM replication - primary/backup approach

1. Pause primary VM (every 25ms) and transmit all changed state (e.g., memory pages) to backup.
2. Backup acknowledges to the primary when complete state has been received.
3. Primary’s buffered network output is released.

Synchronize primary/backup every 25ms

Remus [NSDI’08]
Classic VM replication - primary/backup approach

Remus [NSDI’08]

Synchronize primary/backup every 25ms
1. Pause primary VM (every 25ms) and transmit all changed state (e.g., memory pages) to backup.
2. Backup acknowledges to the primary when complete state has been received.
3. Primary’s buffered network output is released.
Two limitations of primary/backup approach (1)

• Too many memory pages have to be copied and transferred, greatly ballooned client-perceived latency

Redis latency with varied # of clients (4 vCPUs per VM)

<table>
<thead>
<tr>
<th># of concurrent clients</th>
<th>Page transfer size (MB)</th>
</tr>
</thead>
<tbody>
<tr>
<td>16</td>
<td>20.9</td>
</tr>
<tr>
<td>48</td>
<td>68.4</td>
</tr>
<tr>
<td>80</td>
<td>110.5</td>
</tr>
</tbody>
</table>
Two limitations of primary/backup approach (2)

• The split-brain problem

![Diagram showing primary and backup systems with KVS and VMM components.]
Two limitations of primary/backup approach (2)

• The split-brain problem
Two limitations of primary/backup approach (2)

• The split-brain problem
Two limitations of primary/backup approach (2)

• The split-brain problem

<table>
<thead>
<tr>
<th>Primary</th>
<th>Outdated primary</th>
</tr>
</thead>
<tbody>
<tr>
<td>Guest VM</td>
<td>page</td>
</tr>
<tr>
<td>KVS</td>
<td>VMM</td>
</tr>
<tr>
<td>Output buffer</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>New primary</th>
<th>Backup</th>
</tr>
</thead>
<tbody>
<tr>
<td>page</td>
<td>Guest VM</td>
</tr>
<tr>
<td>KVS</td>
<td>VMM</td>
</tr>
</tbody>
</table>

client1: $x=5$
client2: $x=7$
Two limitations of primary/backup approach (2)

- The split-brain problem

![Diagram showing primary and backup VMs with KVS and X values]

- Outdated primary
  - KVS
  - X=5
  - VMM
  - Output buffer

- New primary
  - KVS
  - X=7
  - VMM
Two limitations of primary/backup approach (2)

• The split-brain problem

[Diagram showing the split-brain problem]
State Machine Replication (SMR): Powerful
State Machine Replication (SMR): Powerful
State Machine Replication (SMR): Powerful

client1  client2

backup
primary
backup
State Machine Replication (SMR): Powerful

- SMR systems: Chubby, Zookeeper, Raft [ATC’14], Consensus in a box [NSDI’15], NOPaxos[OSDI’16], APUS [SoCC’17]

Ensure same execution states
State Machine Replication (SMR): Powerful

- SMR systems: Chubby, Zookeeper, Raft [ATC’14], Consensus in a box [NSDI’15], NOPaxos[OSDI’16], APUS [SoCC’17]

- Ensure same execution states
- Strong fault tolerance guarantee without split-brain problem
State Machine Replication (SMR): Powerful

- SMR systems: Chubby, Zookeeper, Raft [ATC’14], Consensus in a box [NSDI’15], NOPaxos[OSDI’16], APUS [SoCC’17]

- Ensure same execution states
- Strong fault tolerance guarantee without split-brain problem

- Need to handle non-determinism
  - Deterministic multithreading (e.g., CRANE [SOSP’15]) - slow
  - Manually annotate service code to capture non-determinism (e.g., Eve [OSDI’12]) - error prone
Making a choice

State machine replication

Pros:
- 🌟 Good performance by ensuring the same execution states
- 🌟 Solve the split-brain problem

Cons:
- 😞 Hard to handle non-determinism

Primary/backup approach

Pros:
- 😊 Automatically handle non-determinism

Cons:
- 😊 Unsatisfactory performance due to transferring large amount of state
- 😞 Have the split-brain problem
PLOVER: Combining SMR and primary/backup

• Simple to achieve by carefully designing the consensus protocol
  • Step 1: Use Paxos to ensure the same total order of requests for replicas
  • Step 2: Invoke VM synchronization periodically and then release replies

• Combines the benefits of SMR and primary/backup
  • Step 1 makes primary/backup have mostly the same memory (up to 97%), then PLOVER need only copy and transfer a small portion of the memory
  • Step 2 automatically addresses non-determinism and ensures external consistency

• Challenges:
  • How to achieve consensus and synchronize VM efficiently?
  • When to do the VM synchronization for primary/backup to maximize the same memory pages?
PLOVER architecture
PLOVER architecture

Primary
- VM page
- Sync VM
- service
- VMM
- log
- Output buffer
- consensus
- Client

Backup
- Sync VM
- page VM
- service
- VMM
- log
- consensus
- Output buffer

Witness
- consensus
RDMA-based input consensus:
- Primary: propose request and execute
- Backup: agree on request and execute
- Witness: agree on request and ignore
PLOVER architecture

RDMA-based input consensus:
- Primary: propose request and execute
- Backup: agree on request and execute
- Witness: agree on request and ignore
PLOVER architecture

RDMA-based input consensus:
- Primary: propose request and execute
- Backup: agree on request and execute
- Witness: agree on request and ignore
RDMA-based input consensus:
- Primary: propose request and execute
- Backup: agree on request and execute
- Witness: agree on request and ignore
PLOVER architecture

RDMA-based input consensus:
- Primary: propose request and execute
- Backup: agree on request and execute
- Witness: agree on request and ignore

RDMA-based VM synchronization:
PLOVER architecture

RDMA-based input consensus:
- Primary: propose request and execute
- Backup: agree on request and execute
- Witness: agree on request and ignore

RDMA-based VM synchronization:
1. Exchange and union dirty page bitmap
2. Compute hash of each dirty page
3. Compare hashes
4. Transfer divergent pages
RDMA-based input consensus:
- Primary: propose request and execute
- Backup: agree on request and execute
- Witness: agree on request and ignore

RDMA-based VM synchronization:
1. Exchange and union dirty page bitmap
2. Compute hash of each dirty page
3. Compare hashes
4. Transfer divergent pages

PLOVER architecture
RDMA-based input consensus:
- Primary: propose request and execute
- Backup: agree on request and execute
- Witness: agree on request and ignore

RDMA-based VM synchronization:
1. Exchange and union dirty page bitmap
2. Compute hash of each dirty page
3. Compare hashes
4. Transfer divergent pages
When to decide VM synchronization period?
When to decide VM synchronization period?
When to decide VM synchronization period?
When to decide VM synchronization period?

Issue of not choosing synchronization timing carefully
• Large amount of divergent state

Synchronize when processing is almost finished!
• CPU and disk usage is almost zero when service finishes processing
• Non-intrusive scheme to monitor service state
• Invoke synchronization when CPU and disk usage is nearly zero
PLOVER addressed other practical challenges

• Concurrent hash computation of dirty pages
• Fast consensus without interrupting the VMM’s I/O event loop
• Collect service running state from VMM without hurting performance
• Full integration with KVM-QEMU
• ...

Evaluation setup

• Three replica machines
  • Dell R430 server
  • Connected with 40Gbps network
  • Guest VM configured with 4 vCPU and 16GB memory

• Metrics: measured both throughput and latency with 95% percentile

• Compared with three state-of-the-art VM fault tolerance systems
  • Remus (NSDI’08): use its latest KVM-based implementation developed by KVM
  • STR (DSN’09) and COLO (SoCC’13): various optimizations of Remus. E.g., COLO skips synchronization if network outputs from two VMs are the same
- Evaluated PLOVER on 12 programs, grouped into 8 services

<table>
<thead>
<tr>
<th>service</th>
<th>Program type</th>
<th>Benchmark</th>
<th>Workload</th>
</tr>
</thead>
<tbody>
<tr>
<td>Redis</td>
<td>Key value store</td>
<td>self</td>
<td>50% SET, 50% GET</td>
</tr>
<tr>
<td>SSDB</td>
<td>Key value store</td>
<td>self</td>
<td>50% SET, 50% GET</td>
</tr>
<tr>
<td>MediaTomb</td>
<td>Multimedia storage server</td>
<td>ApacheBench</td>
<td>Transcoding videos</td>
</tr>
<tr>
<td>pgSQL</td>
<td>Database server</td>
<td>pgbench</td>
<td>TPC-B</td>
</tr>
<tr>
<td>DjCMS (Nginx, Python, MySQL)</td>
<td>Content management system</td>
<td>ApacheBench</td>
<td>Web requests on a dashboard page</td>
</tr>
<tr>
<td>Tomcat</td>
<td>HTTP web server</td>
<td>ApacheBench</td>
<td>Web requests on a shopping store page</td>
</tr>
<tr>
<td>lighttpd</td>
<td>HTTP web server</td>
<td>ApacheBench</td>
<td>Watermark image with PHP</td>
</tr>
<tr>
<td>Node.js</td>
<td>HTTP web server</td>
<td>ApacheBench</td>
<td>Web requests on a messenger bot</td>
</tr>
</tbody>
</table>
Evaluation questions

• How does PLOVER compare to unreplicated VM and state-of-the-art VM fault tolerance systems?

• How does PLOVER scale to multi-core?

• What is PLOVER’s CPU footprint?

• How robust is PLOVER to failures?
  • Handle network partition, leader failure, etc, efficiently

• Comparison of PLOVER and other three systems on different parameter settings?
  • PLOVER is still much faster than the three systems
Throughput on four services

- Unreplicated
- Plover
- COLO
- Remus
- STR
Throughput on the other four services
Lighttpd+PHP performance analysis

### PLOVER:

<table>
<thead>
<tr>
<th>Interval</th>
<th>Dirty Page</th>
<th>Same</th>
<th>Transfer</th>
</tr>
</thead>
<tbody>
<tr>
<td>86ms</td>
<td>33.9K</td>
<td>97%</td>
<td>2.8ms</td>
</tr>
</tbody>
</table>

### Remus:

<table>
<thead>
<tr>
<th>Sync-interval</th>
<th>Dirty Page</th>
<th>Transfer</th>
</tr>
</thead>
<tbody>
<tr>
<td>25ms (Remus-Xen default)</td>
<td>33.3K</td>
<td>53.5ms</td>
</tr>
<tr>
<td>100ms (Remus-KVM default)</td>
<td>33.9K</td>
<td>55.7ms</td>
</tr>
</tbody>
</table>

### Analysis:

PLOVER needs to transfer only 33.9k * 3% = 1.0K pages, but Remus, STR, and COLO need to transfer all or most of the 33K dirty pages. E.g., since most network outputs from two VMs differ, COLO has to do synchronizations for almost every output packet.
pgSQL performance analysis

PLOVER is slower than COLO on pgSQL

- COLO safely skips synchronization because most network outputs from two VMs are the same
Performance Summary (4 vCPU per VM)

- PLOVER’s throughput is 21% lower than unreplicated, 0.9X higher than Remus, 1.0X higher than COLO, 1.4X higher than STR
- 72% ~ 97% dirty memory pages between PLOVER’s primary and backup are the same
- PLOVER’s TCP implementation throughput is still 0.9X higher the three systems on average
Multi-core Scalability (4vCPU - 16vCPU per VM)

- Redis, DjCMS, pgSQL, and Node.js are not listed because they don’t need many vCPUs per VM to improve throughput
  - E.g., Redis is single-threaded
Conclusion and Ongoing Work

• PLOVER: efficiently replicate VM with strong fault tolerance
  • Low performance overhead, scalable to multi-core, robust to replica failures

• Collaborating with Huawei for technology transfer
  • Funded by Huawei Innovation Research Program 2017
  • Submitted a patent (Patent Cooperation Treaty ID: 85714660PCT01)

• https://github.com/hku-systems/plover