Protocol-Aware Recovery for Consensus-Based Storage

Ramnatthan Alagappan, Aishwarya Ganesan, Eric Lee*, Aws Albarghouthi, Vijay Chidambaram*, Andrea Arpaci-Dusseau, and Remzi Arpaci-Dusseau

University of Wisconsin – Madison
*University of Texas at Austin
Redundancy Enables Reliability

Redundancy helps distributed storage systems mask failures
Redundancy Enables Reliability

Redundancy helps distributed storage systems mask failures
Redundancy Enables Reliability

Redundancy helps distributed storage systems mask failures

- system crashes
Redundancy Enables Reliability

Redundancy helps distributed storage systems mask failures
- system crashes
- network failures
Redundancy Enables Reliability

Redundancy helps distributed storage systems mask failures

- system crashes
- network failures

System as a whole unaffected

- data is available
- data is correct
How About Faulty Data?
How About Faulty Data?

Data could be faulty
- corrupted (disk corruption)
- inaccessible (latent errors)
How About Faulty Data?

Data could be faulty
- corrupted (disk corruption)
- inaccessible (latent errors)

We call these storage faults
How to Recover Faulty Data?
How to Recover Faulty Data?

A widely used approach: delete the data on the faulty node and restart it.
How to Recover Faulty Data?

A widely used approach: delete the data on the faulty node and restart it

ZooKeeper fails to start? How can I fix?
Try clearing all the state in Zookeeper: stop Zookeeper, wipe the Zookeeper data directory, restart it

– A top Stackoverflow answer [1]

How to Recover Faulty Data?

A widely used approach: delete the data on the faulty node and restart it.

ZooKeeper fails to start? How can I fix?
Try clearing all the state in Zookeeper: stop Zookeeper, wipe the Zookeeper data directory, restart it.

– A top Stackoverflow answer [1]

A server might not be able to read its database … because of some file corruption in the transaction logs...in such a case, make sure all the other servers in your ensemble are up and working….go ahead and clean the database of the corrupt server. Delete all the files in datadir... Restart the server…

– Recommendation from developers [2]

How to Recover Faulty Data?

A widely used approach: **delete** the data on the faulty node and **restart** it.

- **ZooKeeper** fails to start? How can I fix?
  - Try clearing all the state in Zookeeper: **stop** Zookeeper, wipe the Zookeeper data directory, **restart** it.
  - [A top Stackoverflow answer](https://stackoverflow.com/questions/17038957/) [1]

A server might not be able to read its database ... because of some **file corruption** in the transaction logs...in such a case, make sure all the other servers in your ensemble are up and working....**go ahead and clean the database** of the corrupt server. Delete all the files in datadir... **Restart** the server...

- **Recommendation from developers** [2]

Looks reasonable: redundancy will help.

How to Recover Faulty Data?

A widely used approach: delete the data on the faulty node and restart it.
How to Recover Faulty Data?

A widely used approach: delete the data on the faulty node and restart it.
How to Recover Faulty Data?

A widely used approach: delete the data on the faulty node and restart it.
How to Recover Faulty Data?

A widely used approach: delete the data on the faulty node and restart it.

The approach seems intuitive and works - all good, right?
Unfortunately, No…Not So Easy!
Unfortunately, No…Not So Easy!

Surprisingly, can lead to a global data loss!
Unfortunately, No…Not So Easy!

Surprisingly, can lead to a **global data loss**!

![Diagram showing interconnected databases](image-url)
Unfortunately, No…Not So Easy!

Surprisingly, can lead to a global data loss!
Unfortunately, No…Not So Easy!

Surprisingly, can lead to a global data loss!
Unfortunately, No…Not So Easy!

Surprisingly, can lead to a **global data loss!**
Unfortunately, No…Not So Easy!

Surprisingly, can lead to a **global data loss!**
Unfortunately, No…Not So Easy!

Surprisingly, can lead to a global data loss!
Unfortunately, No…Not So Easy!

Surprisingly, can lead to a global data loss!

This majority has no idea about the committed data.
Unfortunately, No…Not So Easy!

Surprisingly, can lead to a **global data loss**!

This majority has **no idea** about the **committed data**. Committed data is **lost**!
Problem: Approach is Protocol-Oblivious
Problem: Approach is Protocol-Oblivious

The recovery approach is oblivious to the underlying protocols used by the distributed system
The recovery approach is oblivious to the underlying protocols used by the distributed system.

e.g., the delete + rebuild approach was oblivious to the protocol used by the system to update the replicated data.
Our Proposal: Protocol-Aware Recovery (PAR)

To safely recover, a recovery approach should be carefully designed based on properties of underlying protocols of the distributed system.

e.g., is there a dedicated leader? constraints on leader election? how is the replicated state updated? what are the consistency guarantees?

We call such an approach protocol-aware
Our Focus: PAR for Replicated State Machines (RSM)
Our Focus: PAR for Replicated State Machines (RSM)

Why RSM?
Why RSM?

→ most fundamental piece in building reliable distributed systems

Our Focus: PAR for Replicated State Machines (RSM)
Why RSM?

→ most fundamental piece in building reliable distributed systems
→ many systems depend upon RSM

Our Focus: PAR for Replicated State Machines (RSM)
Our Focus: PAR for Replicated State Machines (RSM)

Why RSM?

- most fundamental piece in building reliable distributed systems
- many systems depend upon RSM

- protecting RSM will improve reliability of many systems
Why RSM?

→ most fundamental piece in building reliable distributed systems
→ many systems depend upon RSM

→ protecting RSM will improve reliability of many systems

A hard problem
Our Focus: PAR for Replicated State Machines (RSM)

Why RSM?

→ most fundamental piece in building reliable distributed systems
→ many systems depend upon RSM

→ protecting RSM will improve reliability of many systems

A hard problem

→ strong guarantees, even a small misstep can break guarantees
This Work
This Work

Study popular systems and analyze prior approaches
This Work

Study popular systems and analyze prior approaches

→ approaches in most systems are protocol-oblivious
This Work

Study popular systems and analyze prior approaches

- approaches in most systems are protocol-oblivious
- some use protocol knowledge, but incorrectly
This Work

Study popular systems and analyze prior approaches

- approaches in most systems are **protocol-oblivious**
- some use protocol knowledge, but **incorrectly**
- violate safety (e.g., data loss) or cause **unavailability**
This Work

Study popular systems and analyze prior approaches

- approaches in most systems are protocol-obliviou
- some use protocol knowledge, but incorrectly
- violate safety (e.g., data loss) or cause unavailability

Our solution: CTRL (Corruption-Tolerant RepLication)
This Work

Study popular systems and analyze prior approaches
- approaches in most systems are protocol-oblivious
- some use protocol knowledge, but incorrectly
- violate safety (e.g., data loss) or cause unavailability

Our solution: CTRL (Corruption-Tolerant RepLication)
- a PAR approach, exploits properties of RSM protocols
This Work

Study popular systems and analyze prior approaches
- approaches in most systems are protocol-oblivious
- some use protocol knowledge, but incorrectly
- violate safety (e.g., data loss) or cause unavailability

Our solution: CTRL (Corruption-Tolerant RepLication)
- a PAR approach, exploits properties of RSM protocols
- guarantees safety and high availability with low performance overhead
This Work

Study popular systems and analyze prior approaches

- approaches in most systems are protocol-oblivious
- some use protocol knowledge, but incorrectly
- violate safety (e.g., data loss) or cause unavailability

Our solution: **CTRL** (*Corruption-Tolerant RepLication*)

- a PAR approach, exploits properties of RSM protocols
- guarantees safety and high availability with low performance overhead
- applied to LogCabin and ZooKeeper
This Work

Study popular systems and analyze prior approaches

- approaches in most systems are protocol-oblivious
- some use protocol knowledge, but incorrectly
- violate safety (e.g., data loss) or cause unavailability

Our solution: **CTRL** (Corruption-Tolerant RepLication)

- a PAR approach, exploits properties of RSM protocols
- guarantees safety and high availability with low performance overhead
- applied to LogCabin and ZooKeeper
- experimentally verified guarantees and little overheads (4%-8%)
Outline

Introduction

Replicated state machines

Current approaches to storage faults

CTRL: corruption-tolerant replication

Evaluation

Summary and conclusion
RSM Overview

RSM: a paradigm to make a program/state machine more reliable
RSM Overview

RSM: a paradigm to make a program/state machine more reliable

key idea: run on many servers,
RSM Overview

RSM: a paradigm to make a program/state machine more reliable

key idea: run on many servers, same initial state,
RSM Overview

RSM: a paradigm to make a program/state machine more reliable

key idea: run on many servers, same initial state, same sequence of inputs,
RSM Overview

RSM: a paradigm to make a program/state machine more reliable

key idea: run on many servers, same initial state, same sequence of inputs, will produce same outputs

clients

inputs CBA
RSM Overview

RSM: a paradigm to make a program/state machine more reliable

key idea: run on many servers, same initial state, same sequence of inputs, will produce same outputs

Always correct and available if a majority of servers are functional
RSM Overview

RSM: a paradigm to make a program/state machine more reliable
key idea: run on many servers, same initial state, same sequence of inputs, will produce same outputs

Always correct and available if a majority of servers are functional

A consensus algorithm (e.g., Paxos, Raft, or ZAB) ensures SMs process commands in the same order
Replicated State Update

Disk

Consensus

State Machine

Consensus

State Machine

Consensus

State Machine
Replicated State Update
Replicated State Update
Replicated State Update
Replicated State Update

Leader

Consensus

State Machine

Follower

Consensus

State Machine

Follower

Consensus

State Machine

DISK

Log

Log

Log
Replicated State Update
Replicated State Update

A Consensus State Machine Log DISK

B Consensus Log

C Consensus Log

Leader

Follower

Follower

State Machine

State Machine

State Machine
Replicated State Update

Leader

Consensus

State Machine

DISK

Log

ABCD

Follower

Consensus

State Machine

Log

ABCD

Follower

Consensus

State Machine

Log

ABCD
Replicated State Update
Replicated State Update

apply to SM once majority log the command

Leader

Consensus

State Machine

ACK

ACK

Follower

Consensus

State Machine

Follower

Consensus

State Machine

Follower

Consensus

State Machine
Replicated State Update

apply to SM once majority log the command
Replicated State Update

apply to SM once majority log the command

Leader

Consensus

State Machine

ACK

ACK

Follower

Consensus

State Machine

Follower

Consensus

State Machine

DISK

A

B

C

Log

A

B

C

Log

A

B

C

Log
Replicated State Update

apply to SM once majority log the command

Command is **committed**
Safety condition: C must **not** be lost or overwritten!

ACK

Leader

Consensus

State Machine

DISK

ACK

Log

Follower

ACK

Result

apply to SM once majority log the command

Command is **committed**
Safety condition: C must **not** be lost or overwritten!

ACK

Leader

Consensus

State Machine

DISK

ACK

Log

Follower

ACK

Result

apply to SM once majority log the command

Command is **committed**
Safety condition: C must **not** be lost or overwritten!

ACK

Leader

Consensus

State Machine

DISK

ACK

Log

Follower

ACK

Result
Replicated State Update

apply to SM once majority log the command

Command is **committed**

Safety condition: C must **not** be lost or overwritten!
Replicated State Update

Command is committed
Safety condition: C must not be lost or overwritten!

apply to SM once majority log the command

result

Leader

ACK

ACK

Follower

Follower

Follower
RSM Persistent Structures
RSM Persistent Structures

Log - commands are persistently stored

Snapshots - persistent image of the state machine
RSM Persistent Structures

Log - commands are persistently stored

Snapshots - persistent image of the state machine

Metainfo - critical meta-data structures (e.g., whom did I vote for?)
RSM Persistent Structures

Log - commands are persistently stored

Snapshots - persistent image of the state machine

Metainfo - critical meta-data structures (e.g., whom did I vote for?)

- specific to each node, should not be recovered from redundant copies on other nodes
RSM Persistent Structures

Log - commands are persistently stored

Snapshots - persistent image of the state machine

Metainfo - critical meta-data structures (e.g., whom did I vote for?)

- specific to each node, should not be recovered from redundant copies on other nodes

disk corruption or latent sector errors
RSM Persistent Structures

- **Log** - commands are persistently stored
- **Snapshots** - persistent image of the state machine
- **Metainfo** - critical meta-data structures (e.g., whom did I vote for?)

- disk corruption or latent sector errors

get corrupted data (e.g., ext2/3/4)
get error (e.g., any FS on latent errors, btrfs on a corruption)

Specific to each node, should not be recovered from redundant copies on other nodes.
Outline

Introduction
Replicated state machines
**Current approaches to storage faults**
CTRL: corruption-tolerant replication
Evaluation
Summary and conclusion
Current Approaches to Handling Storage Faults
Current Approaches to Handling Storage Faults

Methodology
  ➔ fault-injection study of practical systems (ZooKeeper, LogCabin, etcd, a Paxos-based system)
  ➔ analyze approaches from prior research
Current Approaches to Handling Storage Faults

Methodology
- fault-injection study of practical systems (ZooKeeper, LogCabin, etcd, a Paxos-based system)
- analyze approaches from prior research

Protocol-oblivious
- do not use any protocol knowledge
Current Approaches to Handling Storage Faults

Methodology
- fault-injection study of practical systems (ZooKeeper, LogCabin, etcd, a Paxos-based system)
- analyze approaches from prior research

Protocol-oblivious
- do not use any protocol knowledge

Protocol-aware
- use some protocol knowledge but incorrectly or ineffectively
Protocol-Oblivious: Crash
Protocol-Oblivious: Crash

Crash

→ use checksums and catch I/O errors
→ crash the node upon detection
→ popular in practical systems
→ safe but poor availability
Protocol-Oblivious: Crash

Crash

- use checksums and catch I/O errors
- crash the node upon detection
- popular in practical systems
- safe but poor availability
Protocol-Oblivious: Crash

Crash

- use checksums and catch I/O errors
- crash the node upon detection
- popular in practical systems
- safe but poor availability
Protocol-Oblivious: Crash

Crash

- use checksums and catch I/O errors
- crash the node upon detection
- popular in practical systems
- safe but poor availability
Protocol-Oblivious: Crash

Crash
- use checksums and catch I/O errors
- crash the node upon detection
- popular in practical systems
- safe but poor availability
Protocol-Oblivious: Crash

Crash
- use checksums and catch I/O errors
- crash the node upon detection
- popular in practical systems
- safe but poor availability
Protocol-Oblivious: Crash

Crash

- use checksums and catch I/O errors
- crash the node upon detection
- popular in practical systems
- safe but poor availability
Protocol-Oblivious: Crash

Crash

- use checksums and catch I/O errors
- crash the node upon detection
- popular in practical systems
- safe but poor availability
Protocol-Oblivious: Crash

Crash
- use checksums and catch I/O errors
- crash the node upon detection
- popular in practical systems
- safe but poor availability

Restarting the node does not help
- persistent fault, so remain in crash-restart loop
- need error-prone manual intervention (can lead to safety violations)
Protocol-Oblivious: Truncate
Protocol-Oblivious: Truncate

Truncate
Protocol-Oblivious: Truncate

Truncate

→ truncate “faulty” portions upon detection

detect using checksums
Protocol-Oblivious: Truncate

Truncate

→ truncate “faulty” portions upon detection

However, can lead to safety violations
Protocol-Oblivious: Truncate

**Truncate**

→ truncate “faulty” portions upon detection

However, can lead to **safety violations**

S1

S2

S3

S4

S5

S2 - Leader
A,B,C

committed

detect using checksums

A

C

A
Protocol-Oblivious: Truncate

Truncate

→ truncate “faulty” portions upon detection

However, can lead to safety violations

A, B, C committed

S2 - Leader

A, B, C corrupted at S1
Protocol Oblivious: Truncate

**Truncate**

→ truncate “faulty” portions upon detection

However, can lead to safety violations

S1

\[
\begin{array}{c|c|c|c}
S1 & A & B & C \\
S2 & A & B & C \\
S3 & A & B & C \\
S4 & & & \\
S5 & & & \\
\end{array}
\]

S2 - Leader A,B,C committed

S2

\[
\begin{array}{c|c|c|c}
S2 & \text{A, B, C} \\
S2 & A & B & C \\
S2 & A & B & C \\
S2 & & & \\
S2 & & & \\
\end{array}
\]

Entry A corrupted at S1

S2

\[
\begin{array}{c|c|c|c}
S2 & & & \\
S2 & & & \\
S2 & & & \\
S2 & & & \\
S2 & A & B & C \\
\end{array}
\]

truncates faulty and all subsequent entries

detect using checksums

A
Protocol-Oblivious: Truncate

**Truncate**

→ truncate “faulty” portions upon detection

However, can lead to **safety violations**

\[\text{S1} \rightarrow \text{S2} \rightarrow \text{S3} \rightarrow \text{S4} \rightarrow \text{S5} \]

- **S2** - Leader
- A, B, C committed
- **S2** - Entry A corrupted at S1
- **S2** - truncates faulty and all subsequent entries
- **S2, S3** crash; **S1, S4, S5** form a majority
- **S1** - Leader

detect using checksums
Protocol-Oblivious: Truncate

**Truncate**

- truncate “faulty” portions upon detection

However, can lead to **safety violations**

S1

ABC

S2

ABC

S3

ABC

S4

B

S5

A

S2 - Leader A,B,C committed

Entry A corrupted at S1

truncates faulty and all subsequent entries

S2, S3 crash; S1, S4, S5 form a majority

A,B,C silently lost!
Protocol-Oblivious: Truncate

---

**Truncate**

→ truncate “faulty” portions upon detection

However, can lead to **safety violations**

S2 - Leader
A,B,C
committed

Entry A
corrupted at S1

truncates
faulty and all
subsequent
entries

S2, S3 crash; S1, S4, S5 form a majority

S1 - Leader

A,B,C silently lost!
Protocol-Oblivious: Truncate

**Truncate**

- truncate "faulty" portions upon detection

However, can lead to safety violations

- detect using checksums

S2 - Leader A,B,C committed

Entry A corrupted at S1

truncates faulty and all subsequent entries

S2, S3 crash; S1, S4, S5 form a majority

S1 - Leader

A,B,C silently lost!

S2, S3 follow leader's log, removing A,B,C
Recovery Approaches Summary

<table>
<thead>
<tr>
<th>Class</th>
<th>Protocol-oblivial</th>
<th>Protocol-aware</th>
</tr>
</thead>
<tbody>
<tr>
<td>Protocol-oblivial</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Protocol-aware</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
## Recovery Approaches Summary

<table>
<thead>
<tr>
<th>Class</th>
<th>Approach</th>
</tr>
</thead>
<tbody>
<tr>
<td>Protocol-oblivious</td>
<td>NoDetection</td>
</tr>
<tr>
<td></td>
<td>Crash</td>
</tr>
<tr>
<td></td>
<td>Truncate</td>
</tr>
<tr>
<td></td>
<td>DeleteRebuild</td>
</tr>
<tr>
<td>Protocol-aware</td>
<td>MarkNonVote [1]</td>
</tr>
<tr>
<td></td>
<td>Reconfigure [2]</td>
</tr>
<tr>
<td></td>
<td>Byzantine FT</td>
</tr>
</tbody>
</table>

## Recovery Approaches Summary

<table>
<thead>
<tr>
<th>Class</th>
<th><strong>Approach</strong></th>
<th>Safety</th>
</tr>
</thead>
<tbody>
<tr>
<td>Protocol-oblivioues</td>
<td>NoDetection Crash Truncate DeleteRebuild</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Class</th>
<th>Approach</th>
<th>Safety</th>
<th>Availability</th>
</tr>
</thead>
<tbody>
<tr>
<td>Protocol-oblivious</td>
<td>NoDetection</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Crash</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Truncate</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>DeleteRebuild</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Protocol-aware</td>
<td>MarkNonVote [1]</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Reconfigure [2]</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Byzantine FT</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

# Recovery Approaches Summary

<table>
<thead>
<tr>
<th>Class</th>
<th>Approach</th>
<th>Safety</th>
<th>Availability</th>
<th>Perform - ance</th>
<th>No intervention</th>
<th>No extra nodes</th>
<th>Fast recovery</th>
<th>Low complexity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Protocol- oblivious</td>
<td>NoDetection Crash Truncate DeleteRebuild</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

## Recovery Approaches Summary

<table>
<thead>
<tr>
<th>Class</th>
<th>Approach</th>
<th>Safety</th>
<th>Availability</th>
<th>Performance</th>
<th>No intervention</th>
<th>No extra nodes</th>
<th>Fast recovery</th>
<th>Low complexity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Protocol-oblivious</td>
<td>NoDetection</td>
<td>☒</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>NA</td>
<td>☑</td>
</tr>
<tr>
<td></td>
<td>Crash</td>
<td>☑</td>
<td>☒</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>NA</td>
<td>☑</td>
</tr>
<tr>
<td></td>
<td>Truncate</td>
<td>☒</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>NA</td>
<td>☑</td>
</tr>
<tr>
<td></td>
<td>DeleteRebuild</td>
<td>☒</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>NA</td>
<td>☑</td>
</tr>
<tr>
<td>Protocol-aware</td>
<td>MarkNonVote [1]</td>
<td>☒</td>
<td>☒</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>NA</td>
<td>☑</td>
</tr>
<tr>
<td></td>
<td>Reconfigure [2]</td>
<td>☑</td>
<td>☒</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
</tr>
<tr>
<td></td>
<td>Byzantine FT</td>
<td>☑</td>
<td>☒</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
</tr>
</tbody>
</table>

## Recovery Approaches Summary

<table>
<thead>
<tr>
<th>Class</th>
<th>Approach</th>
<th>Safety</th>
<th>Availability</th>
<th>Perform -ance</th>
<th>No intervention</th>
<th>No extra nodes</th>
<th>Fast recovery</th>
<th>Low complexity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Protocol- oblivious</td>
<td>NoDetection</td>
<td>🚧</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
</tr>
<tr>
<td></td>
<td>Crash</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
</tr>
<tr>
<td></td>
<td>Truncate</td>
<td>🚧</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
</tr>
<tr>
<td></td>
<td>DeleteRebuild</td>
<td>🚧</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
</tr>
<tr>
<td>Protocol- aware</td>
<td>MarkNonVote [1]</td>
<td>✔️</td>
<td>🚧</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
</tr>
<tr>
<td></td>
<td>Reconfigure [2]</td>
<td>✔️</td>
<td>🚧</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
</tr>
<tr>
<td></td>
<td>Byzantine FT</td>
<td>🚧</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Class</th>
<th>Approach</th>
<th>Safety</th>
<th>Availability</th>
<th>Performance</th>
<th>No intervention</th>
<th>No extra nodes</th>
<th>Fast recovery</th>
<th>Low complexity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Protocol-oblivious</td>
<td>NoDetection</td>
<td></td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>NA</td>
<td>✔️</td>
</tr>
<tr>
<td></td>
<td>Crash</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>✔️</td>
</tr>
<tr>
<td></td>
<td>Truncate</td>
<td>❌</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>NA</td>
<td>✔️</td>
</tr>
<tr>
<td></td>
<td>DeleteRebuild</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>NA</td>
<td>✔️</td>
</tr>
<tr>
<td>Protocol-aware</td>
<td>MarkNonVote [1]</td>
<td></td>
<td></td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>NA</td>
<td>✔️</td>
</tr>
<tr>
<td></td>
<td>Reconfigure [2]</td>
<td></td>
<td></td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>NA</td>
<td>✔️</td>
</tr>
<tr>
<td></td>
<td>Byzantine FT</td>
<td></td>
<td></td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>NA</td>
<td>✔️</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Class</th>
<th>Approach</th>
<th>Safety</th>
<th>Availability</th>
<th>Perform-ance</th>
<th>No intervention</th>
<th>No extra nodes</th>
<th>Fast recovery</th>
<th>Low complexity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Protocol-oblusive</td>
<td>NoDetection</td>
<td>☒</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>NA</td>
<td>✓</td>
</tr>
<tr>
<td></td>
<td>Crash</td>
<td>✓</td>
<td>☒</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>NA</td>
<td>✓</td>
</tr>
<tr>
<td></td>
<td>Truncate</td>
<td>☒</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>NA</td>
<td>✓</td>
</tr>
<tr>
<td></td>
<td>DeleteRebuild</td>
<td>☒</td>
<td>✓</td>
<td>✓</td>
<td>☒</td>
<td>✓</td>
<td>NA</td>
<td>✓</td>
</tr>
<tr>
<td>Protocol-aware</td>
<td>MarkNonVote [1]</td>
<td>☒</td>
<td>☒</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>NA</td>
<td>✓</td>
</tr>
<tr>
<td></td>
<td>Reconfigure [2]</td>
<td>☒</td>
<td>☒</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>NA</td>
<td>✓</td>
</tr>
<tr>
<td></td>
<td>Byzantine FT</td>
<td>✓</td>
<td>☒</td>
<td>✓</td>
<td>☒</td>
<td>✓</td>
<td>NA</td>
<td>✓</td>
</tr>
</tbody>
</table>

## Recovery Approaches Summary

<table>
<thead>
<tr>
<th>Class</th>
<th>Approach</th>
<th>Safety</th>
<th>Availability</th>
<th>Performance</th>
<th>No intervention</th>
<th>No extra nodes</th>
<th>Fast recovery</th>
<th>Low complexity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Protocol-ooblusive</td>
<td>NoDetection</td>
<td>x</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>NA</td>
<td>✓</td>
</tr>
<tr>
<td></td>
<td>Crash</td>
<td>✓</td>
<td>x</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>NA</td>
<td>✓</td>
</tr>
<tr>
<td></td>
<td>Truncate</td>
<td>x</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>X</td>
<td>✓</td>
</tr>
<tr>
<td></td>
<td>DeleteRebuild</td>
<td>x</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>X</td>
<td>✓</td>
</tr>
<tr>
<td>Protocol-aware</td>
<td>MarkNonVote [1]</td>
<td>x</td>
<td>x</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>X</td>
<td>✓</td>
</tr>
<tr>
<td></td>
<td>Reconfigure [2]</td>
<td>✓</td>
<td>x</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>X</td>
<td>✓</td>
</tr>
<tr>
<td></td>
<td>Byzantine FT</td>
<td>✓</td>
<td>x</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>X</td>
<td>✓</td>
</tr>
<tr>
<td>CTRL</td>
<td></td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>

Outline

Introduction

Replicated state machines

Current approaches to storage faults

CTRL: Corruption-tolerant replication

- fault model and guarantees
- local storage layer
- distributed recovery

Evaluation

Summary and conclusion
CTRL Overview
CTRL Overview

Two components
CTRL Overview

Two components

Local storage layer

Storage Layer

manage local data; detect faults
CTRL Overview

Two components
Local storage layer
Distributed recovery

Distributed Recovery

Storage Layer

recover from redundant copies
manage local data; detect faults
CTRL Overview

Two components
Local storage layer
Distributed recovery

Storage Layer
- manage local data; detect faults

Distributed Recovery
- recover from redundant copies
CTRL Overview

Two components
Local storage layer
Distributed recovery

Exploit RSM knowledge to correctly and quickly recover faulty data
CTRL Fault Model
CTRL Fault Model

Standard failure assumptions

→ crashes
→ network failures
CTRL Fault Model

Standard failure assumptions
  ➔ crashes
  ➔ network failures

Augment with storage faults
CTRL Fault Model

Standard failure assumptions

- crashes
- network failures

Augment with **storage faults**

- **data blocks** of log, snapshots, and metainfo can be faulty
  - depending on FS, return corrupted data or turn into errors
CTRL Fault Model

Standard failure assumptions

- crashes
- network failures

Augment with **storage faults**

- **data blocks** of log, snapshots, and metainfo can be faulty
  - depending on FS, return corrupted data or turn into errors
- **FS metadata** blocks could also be faulty
  - e.g., inode of a log file corrupted
  - e.g., files/directories implementing the log may go missing
  - e.g., files may appear with fewer or more bytes
CTRL Guarantees
CTRL Guarantees

Committed data will never be lost

⇒ as long as one intact copy of a data item exists
⇒ correctly remain unavailable when all copies are faulty
CTRL Guarantees

**Committed data will never be lost**
- as long as one intact copy of a data item exists
- correctly remain unavailable when all copies are faulty

Provide the **highest possible availability**
CTRL Local Storage
CTRL Local Storage

Main function: detect and identify
- whether log/snapshot/metainfo faulty or not?
- what is corrupted? (e.g., which log entry?)
CTRL Local Storage

Main function: detect and identify
- whether log/snapshot/metainfo faulty or not?
- what is corrupted? (e.g., which log entry?)

Requirements
- low performance overheads
- low space overheads
CTRL Local Storage

Main function: detect and identify
- whether log/snapshot/metainfo faulty or not?
- what is corrupted? (e.g., which log entry?)

Requirements
- low performance overheads
- low space overheads

An interesting problem: disentangling crashes and corruptions in log
- checksum mismatch due to crash or disk corruption?
Crash-Corruption Entanglement in the Log
Crash-Corruption Entanglement in the Log
Crash-Corruption Entanglement in the Log

append()
Crash-Corruption Entanglement in the Log
Crash-Corruption Entanglement in the Log
Crash-Corruption Entanglement in the Log

Crash during append
  → recovery: can truncate entry - unacknowledged
Crash-Corruption Entanglement in the Log

Crash during append

→ recovery: can truncate entry - unacknowledged
Crash-Corruption Entanglement in the Log

Crash during append
→ recovery: can truncate entry - unacknowledged
Crash-Corruption Entanglement in the Log

Crash during append

→ recovery: can truncate entry - unacknowledged
Crash-Corruption Entanglement in the Log

Crash during append
→ recovery: can truncate entry - unacknowledged

disk corruption
Crash-Corruption Entanglement in the Log

Crash during append
- recovery: can truncate entry - unacknowledged

Disk corruption
- cannot truncate, may lose possibly committed data!
Crash-Corruption Entanglement in the Log

Crash during append
- recovery: can truncate entry - unacknowledged

Disk corruption
- cannot truncate, may lose possibly committed data!

Current systems conflate the two conditions – always truncate
Crash-Corruption Entanglement in the Log

Crash during append
  → recovery: can truncate entry - unacknowledged

Disk corruption
  → cannot truncate, may lose possibly committed data!

Current systems conflate the two conditions – always truncate

CTRL: modified local update – write additional information
  → enables disentanglement, performant - more details in the paper…
CTRL Distributed Recovery

Distributed Log Recovery

Distributed Snapshot Recovery
CTRL Distributed Recovery

Distributed Log Recovery

Distributed Snapshot Recovery

Storage Layer
Properties of Practical Consensus Protocols
Properties of Practical Consensus Protocols

Leader-based

→ single node acts as leader; all updates flow through the leader
Properties of Practical Consensus Protocols

**Leader-based**
- single node acts as leader; all updates flow through the leader

**Epochs**
- a slice of time; only one leader per slice/epoch
- a log entry is uniquely qualified by its index and epoch
Properties of Practical Consensus Protocols

Leader-based
- single node acts as leader; all updates flow through the leader

Epochs
- a slice of time; only one leader per slice/epoch
- a log entry is uniquely qualified by its index and epoch

Leader completeness
- leader guaranteed to have all committed data
Properties of Practical Consensus Protocols

**Leader-based**
- single node acts as leader; all updates flow through the leader

**Epochs**
- a slice of time; only one leader per slice/epoch
- a log entry is uniquely qualified by its index and epoch

**Leader completeness**
- leader guaranteed to have all committed data

Applies to Raft, ZAB, and most implementations of Paxos
Properties of Practical Consensus Protocols

Leader-based
- single node acts as leader; all updates flow through the leader

Epochs
- a slice of time; only one leader per slice/epoch
- a log entry is uniquely qualified by its index and epoch

Leader completeness
- leader guaranteed to have all committed data

Applies to Raft, ZAB, and most implementations of Paxos
CTRL exploits these properties to perform recovery
Follower Log Recovery
Follower Log Recovery

Decouple follower and leader recovery
Follower Log Recovery

Decouple follower and leader recovery

Fixing followers is simple: can be fixed by leader because the leader is guaranteed to have all committed data!
Follower Log Recovery

Decouple follower and leader recovery

Fixing followers is simple: can be fixed by leader because the leader is guaranteed to have all committed data!
Follower Log Recovery

**Decouple** follower and leader recovery

Fixing followers is simple: can be fixed by leader because the leader is **guaranteed to have all committed data!**

Leader:

```
ABC
ABC
ABC
```

Followers:

```
ABC
ABC
ABC
ABC
```

- **index = 2**
- **epoch = e**
Follower Log Recovery

**Decouple** follower and leader recovery

Fixing followers is simple: can be fixed by leader because the leader is guaranteed to have all committed data!
Follower Log Recovery

Decouple follower and leader recovery

Fixing followers is simple: can be fixed by leader because the leader is guaranteed to have all committed data!
Follower Log Recovery

Decouple follower and leader recovery

Fixing followers is simple: can be fixed by leader because the leader is guaranteed to have all committed data!
Follower Log Recovery

**Decouple** follower and leader recovery

Fixing followers is simple: can be fixed by leader because the leader is **guaranteed to have all committed data!**
**Follower Log Recovery**

**Decouple** follower and leader recovery

Fixing followers is simple: can be fixed by leader because the leader is **guaranteed to have all committed data!**
Leader Log Recovery
Leader Log Recovery

Fixing the leader is the tricky part
Leader Log Recovery

Fixing the leader is the tricky part
First, a simple case: some follower has the entry intact
Leader Log Recovery

Fixing the leader is the tricky part
First, a simple case: some follower has the entry intact

Leader

A B
AB
AC
B
AC
Leader Log Recovery

Fixing the leader is the tricky part
First, a simple case: some follower has the entry intact
Leader Log Recovery

Fixing the leader is the tricky part
First, a simple case: some follower has the entry intact

Leader Log Recovery

Leader

index = 3
epoch = e
Leader Log Recovery

Fixing the leader is the tricky part
First, a simple case: some follower has the entry intact

Leader

index = 3
epoch = e
Leader Log Recovery

Fixing the leader is the tricky part
First, a simple case: some follower has the entry intact

Leader

\[
\begin{array}{ccc}
A & B & \text{X} \\
A & B & \text{X} \\
A & C & \text{X} \\
\hline
B & \text{X} & \text{X} \\
A & C & \text{X} \\
\end{array}
\]

index = 3
epoch = e

\[
\begin{array}{ccc}
A & B & C \\
A & B & C \\
A & C & \text{X} \\
\hline
B & \text{X} & \text{X} \\
A & C & \text{X} \\
\end{array}
\]

\[
\begin{array}{ccc}
A & B & \text{X} \\
A & B & \text{X} \\
A & C & \text{X} \\
\hline
B & \text{X} & \text{X} \\
A & C & \text{X} \\
\end{array}
\]
Leader Log Recovery

Fixing the leader is the tricky part
First, a simple case: some follower has the entry intact

Leader Log Recovery

\[
\text{index} = 3 \\
\text{epoch} = e
\]
Fixing the leader is the tricky part
First, a simple case: some follower has the entry intact

Leader Log Recovery
Leader Log Recovery: Determining Commitment
Leader Log Recovery: Determining Commitment

However, sometimes cannot easily recover the leader’s log
Leader Log Recovery: Determining Commitment

However, sometimes cannot easily recover the leader’s log.
Leader Log Recovery: Determining Commitment

However, sometimes cannot easily recover the leader’s log

Leader Log Recovery Diagram:

- Sample log entries for a leader and followers:
  - Leader Log: A B A B A B A B A B
  - Followers Log: A B A B A B A B A B

Diagram illustrating the process of leader log recovery.
Leader Log Recovery: Determining Commitment

However, sometimes cannot easily recover the leader’s log

Main insight: separate committed from uncommitted entries
Leader Log Recovery: Determining Commitment

However, sometimes cannot easily recover the leader’s log

Main insight: separate committed from uncommitted entries
- must fix committed, while uncommitted can be safely discarded
Leader Log Recovery: Determining Commitment

However, sometimes cannot easily recover the leader’s log

Main insight: *separate committed from uncommitted* entries

- must fix committed, while uncommitted can be safely discarded
- discard uncommitted as early as possible for improved availability
Leader Log Recovery: Determining Commitment
Leader Log Recovery: Determining Commitment

Leader queries for a faulty entry
Leader queries for a faulty entry
  ─ if majority say they don’t have the entry → must be an uncommitted entry – can discard and continue
Leader Log Recovery: Determining Commitment

Leader queries for a faulty entry

→ if majority say they don’t have the entry → must be an uncommitted entry – can discard and continue
Leader queries for a faulty entry

→ if majority say they don’t have the entry → must be an uncommitted entry – can discard and continue
Leader Log Recovery: Determining Commitment

Leader queries for a faulty entry

- if majority say they don't have the entry → must be an uncommitted entry – can discard and continue
Leader Log Recovery: Determining Commitment

Leader queries for a faulty entry

→ if majority say they don’t have the entry → must be an uncommitted entry – can discard and continue

→ if committed then at least one node in the majority would have the entry – can fix using that response
Leader Log Recovery: Determining Commitment

Leader queries for a faulty entry

- if majority say they don’t have the entry → must be an uncommitted entry – can discard and continue
- if committed then at least one node in the majority would have the entry – can fix using that response
Leader Log Recovery: Determining Commitment

Leader queries for a faulty entry

→ if majority say they don’t have the entry → must be an uncommitted entry – can discard and continue

→ if committed then at least one node in the majority would have the entry – can fix using that response

discard faulty, continue
don’t have

fix using a response (will get at least one correct response because it is committed)
Leader Log Recovery: Determining Commitment

Leader queries for a faulty entry

→ if majority say they don’t have the entry → must be an uncommitted entry – can discard and continue

→ if committed then at least one node in the majority would have the entry – can fix using that response

discard faulty, continue

fix using a response (will get at least one correct response because it is committed)
Leader Log Recovery: Determining Commitment

Leader queries for a faulty entry
- if majority say they don’t have the entry → must be an uncommitted entry – can discard and continue
- if committed then at least one node in the majority would have the entry – can fix using that response

discard faulty, continue

fix using a response (will get at least one correct response because it is committed)
Leader Log Recovery: Determining Commitment

Leader queries for a faulty entry
- if majority say they don’t have the entry → must be an uncommitted entry – can discard and continue
- if committed then at least one node in the majority would have the entry – can fix using that response
Leader queries for a faulty entry

- if majority say they don’t have the entry → must be an uncommitted entry – can discard and continue
- if committed then at least one node in the majority would have the entry – can fix using that response

Leader Log Recovery: Determining Commitment

don’t have

discard faulty, continue

fix using a response (will get at least one correct response because it is committed)

either fix log or discard, depending on order
Leader Log Recovery: Determining Commitment

Leader queries for a faulty entry
- if majority say they don’t have the entry → must be an uncommitted entry – can discard and continue
- if committed then at least one node in the majority would have the entry – can fix using that response

discard faulty, continue

fix using a response (will get at least one correct response because it is committed)

either fix log or discard, depending on order

1 before 2 - fix
Leader Log Recovery: Determining Commitment

Leader queries for a faulty entry

慎重 if majority say they don’t have the entry → must be an uncommitted entry – can discard and continue
慎重 if committed then at least one node in the majority would have the entry – can fix using that response

discard faulty, continue

fix using a response (will get at least one correct response because it is committed)

either fix log or discard, depending on order
Leader Log Recovery: Determining Commitment

Leader queries for a faulty entry

- If majority say they don’t have the entry, must be an uncommitted entry – can discard and continue
- If committed then at least one node in the majority would have the entry – can fix using that response

29
More In The Paper…
More In The Paper…

Log recovery
- faulty entry on follower unknown to leader
- nodes could be down during recovery
- different entries at same log index

Snapshot recovery
Metainfo recovery
FS metadata fault handling
Outline

Introduction
Replicated state machines
Current approaches to storage faults
CTRL: Corruption-tolerant replication
Evaluation
Summary and conclusion
Evaluation

We apply CTRL in two systems

LogCabin
  - based on Raft

ZooKeeper
  - based on ZAB
Reliability Experiments Example
Reliability Experiments Example
Reliability Experiments Example

file-system data blocks
corruptions
errors

log

A B C
D
Reliability Experiments Example

file-system data blocks

Original

→ corruptions: 30% unsafe or unavailable
→ errors: 50% unavailable
Reliability Experiments Example

file-system data blocks

Original
- corruptions: 30% unsafe or unavailable
- errors: 50% unavailable

CTRL
- corruptions and errors: always safe and available
Reliability Experiments Summary

FS data blocks

Log
Reliability Experiments Summary

FS data blocks

Targeted entries
Reliability Experiments Summary

FS data blocks

Targeted entries

all possible combinations (for thoroughness)
Reliability Experiments Summary

FS data blocks

Targeted entries

Lagging and crashed

A D
A B D
A B C

A B C
C
A

all possible combinations (for thoroughness)

A B C
A B C
A

Log
Reliability Experiments Summary

FS data blocks

Targeted entries

Lagging and crashed

all possible combinations (for thoroughness)
Reliability Experiments Summary

FS data blocks

Targeted entries

Lagging and crashed

all possible combinations (for thoroughness)

Log

Snapshots
Reliability Experiments Summary

**FS data blocks**

![Log and snapshots with labeled data blocks]

**Targeted entries**

- All possible combinations (for thoroughness)

**Lagging and crashed**

![Lagging and crashed data blocks]
Reliability Experiments Summary

Log

FS data blocks

Targeted entries
all possible combinations (for thoroughness)

Lagging and crashed

Snapshots
Reliability Experiments Summary

Log

FS data blocks

Targeted entries

Lagging and crashed

Snapshots

FS Metadata

Faults

FS data blocks:

- A
- B
- C
- D

Targeted entries:

- A
- B
- C

Lagging and crashed:

- A
- B
- C

All possible combinations (for thoroughness):

- A
- B
- C

Faults:

- Snapshots
- FS Metadata
Reliability Experiments Summary

- **Log**
  - FS data blocks
  - Targeted entries
    - all possible combinations (for thoroughness)
  - Lagging and crashed

- **Snapshots**
  - FS data blocks
  - Targeted entries
    - all possible combinations (for thoroughness)
  - Lagging and crashed

- **FS Metadata**
  - Faults
    - Un-openable files
    - Missing files
    - Improper sizes
Reliability Experiments Summary

Log

FS data blocks

Targeted entries

Lagging and crashed

Snapshots

FS Metadata

Faults

Un-openable files

Missing files

Improper sizes
Reliability Results Summary
Reliability Results Summary

Original systems
Reliability Results Summary

Original systems
  → unsafe or unavailable in many cases
Reliability Results Summary

Original systems
- unsafe or unavailable in many cases

CTRL versions
Reliability Results Summary

Original systems
- unsafe or unavailable in many cases

CTRL versions
- safe always and highly available
Reliability Results Summary

Original systems
- unsafe or unavailable in many cases

CTRL versions
- safe always and highly available
- correctly unavailable in some cases (when all copies are faulty)
Update Performance (SSD)
Update Performance (SSD)

Workload: insert entries (1K) repeatedly, background snapshots
Update Performance (SSD)

Workload: insert entries (1K) repeatedly, background snapshots

LogCabin

Throughput (ops/s)

# Clients

ZooKeeper

Throughput (ops/s)

# Clients

Original  CTRL  Original  CTRL
Update Performance (SSD)

Workload: insert entries (1K) repeatedly, background snapshots

Throughput (ops/s)

### LogCabin

<table>
<thead>
<tr>
<th># Clients</th>
<th>Original</th>
<th>CTRL</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td></td>
<td></td>
</tr>
<tr>
<td>16</td>
<td></td>
<td></td>
</tr>
<tr>
<td>32</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### ZooKeeper

<table>
<thead>
<tr>
<th># Clients</th>
<th>Original</th>
<th>CTRL</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td></td>
<td></td>
</tr>
<tr>
<td>16</td>
<td></td>
<td></td>
</tr>
<tr>
<td>32</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Update Performance (SSD)

Workload: insert entries (1K) repeatedly, background snapshots

- LogCabin
- ZooKeeper

Throughput (ops/s)

<table>
<thead>
<tr>
<th># Clients</th>
<th>Original</th>
<th>CTRL</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>1000</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>2000</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>3000</td>
<td></td>
</tr>
<tr>
<td>16</td>
<td>4000</td>
<td></td>
</tr>
<tr>
<td>32</td>
<td>5000</td>
<td></td>
</tr>
</tbody>
</table>

Throughput (ops/s)

<table>
<thead>
<tr>
<th># Clients</th>
<th>Original</th>
<th>CTRL</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>5000</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>10000</td>
<td></td>
</tr>
<tr>
<td>16</td>
<td>15000</td>
<td></td>
</tr>
<tr>
<td>32</td>
<td>20000</td>
<td></td>
</tr>
</tbody>
</table>
Update Performance (SSD)

Workload: insert entries (1K) repeatedly, background snapshots

Overheads (because CTRL’s storage layer writes additional information for each log entry) – however, little: SSDs 4% worst case, disks: 8% to 10%
Update Performance (SSD)

Workload: insert entries (1K) repeatedly, background snapshots

Overheads (because CTRL’s storage layer writes additional information for each log entry) – however, little: SSDs 4% worst case, disks: 8% to 10%

Note: all writes, so worst-case overheads
Summary
Summary

Recovering from storage faults correctly in a distributed system is surprisingly tricky
Summary

Recovering from storage faults correctly in a distributed system is surprisingly tricky.

Most existing recovery approaches are protocol-oblivious – they cause unsafety and low availability.
Summary

Recovering from storage faults correctly in a distributed system is surprisingly tricky.

Most existing recovery approaches are protocol-oblivious – they cause unsafety and low availability.

To correctly and quickly recover, an approach needs to be protocol-aware.
Summary

Recovering from storage faults correctly in a distributed system is surprisingly tricky.

Most existing recovery approaches are protocol-oblivious – they cause unsafety and low availability.

To correctly and quickly recover, an approach needs to be protocol-aware.

CTRL: a protocol-aware recovery approach for RSM

- guarantees safety and provides high availability, with little performance overhead.
Conclusions
Conclusions

Obvious things we take for granted in distributed systems: redundant copies will help recover bad data or redundancy $\Rightarrow$ reliability are surprisingly hard to achieve [1]

[1] Redundancy Does Not Imply Fault Tolerance - Ganesan et al., at FAST ‘17
Conclusions

Obvious things we take for granted in distributed systems: redundant copies will help recover bad data or redundancy $\rightarrow$ reliability are surprisingly hard to achieve [1]

Protocol-awareness is key to use redundancy correctly to recover bad data
  $\rightarrow$ need to be aware of what’s going on underneath in the system

[1] Redundancy Does Not Imply Fault Tolerance - Ganesan et al., at FAST ‘17
Conclusions

Obvious things we take for granted in distributed systems: **redundant copies will help recover bad data** or redundancies $\Rightarrow$ reliability are surprisingly hard to achieve [1]

**Protocol-awareness** is key to use redundancy correctly to recover bad data

- need to be aware of what’s going on underneath in the system

However, only a first step: we have applied PAR only to RSM

- other classes of systems (e.g., quorum-based systems) remain vulnerable

[1] Redundancy Does Not Imply Fault Tolerance - Ganesan et al., at FAST ‘17
Conclusions

Obvious things we take for granted in distributed systems: redundant copies will help recover bad data or redundancy $\rightarrow$ reliability are surprisingly hard to achieve [1].

Protocol-awareness is key to use redundancy correctly to recover bad data

- need to be aware of what’s going on underneath in the system

However, only a first step: we have applied PAR only to RSM

- other classes of systems (e.g., quorum-based systems) remain vulnerable

http://research.cs.wisc.edu/adsl/Publications/par/

[1] Redundancy Does Not Imply Fault Tolerance - Ganesan et al., at FAST ‘17