Fault-Tolerance, Fast and Slow: Exploiting Failure Asynchrony in Distributed Systems

Ramnatthan Alagappan, Aishwarya Ganesan, Jing Liu, Andrea Arpaci-Dusseau, and Remzi Arpaci-Dusseau
Replication Protocols

- Paxos
- Viewstamped replication
- Raft
Replication Protocols

Foundation upon which datacenter systems are built

GFS
Colossus
BigTable

Paxos
Viewstamped replication
Raft
The Two Different Worlds of Replication
The Two Different Worlds of Replication

World-1

|   |

|   |

World-2
The Two Different Worlds of Replication

How and where to store system state?

World-1

World-2
The Two Different Worlds of Replication

How and where to store system state?

World-1

- Disk-durable
  - synchronously persist updates to disks

World-2
The Two Different Worlds of Replication

How and where to store system state?

World-1
- Disk-durable
  - synchronously persist updates to disks

World-2
- Memory-durable
  - buffer updates only in volatile memory
The Two Different Worlds of Replication

How and where to **store system state?**

World-1

- **Disk-durable**
  - synchronously persist updates to disks
  
  Paxos, Raft [ATC ‘14], ZAB [DSN ‘11],
  Gaios [NSDI ‘11], ZooKeeper, etcd, LogCabin …

World-2

- **Memory-durable**
  - buffer updates only in volatile memory
The Two Different Worlds of Replication

How and where to store system state?

World-1

Disk-durable

- synchronously persist updates to disks

- Paxos, Raft [ATC ‘14], ZAB [DSN ‘11], Gaios [NSDI ‘11], ZooKeeper, etcd, LogCabin …

World-2

Memory-durable

- buffer updates only in volatile memory

- Viewstamped replication, NOPaxos [OSDI ‘16], SpecPaxos [NSDI ‘15] …
The Two Different Worlds of Replication

How and where to store system state?

<table>
<thead>
<tr>
<th>World-1</th>
<th>World-2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Disk-durable</td>
<td>Memory-durable</td>
</tr>
<tr>
<td>synchronously persist updates to disks</td>
<td>buffer updates only in volatile memory</td>
</tr>
</tbody>
</table>

Paxos, Raft [ATC ‘14], ZAB [DSN ‘11], Gaios [NSDI ‘11], ZooKeeper, etcd, LogCabin …

Viewstamped replication, NOPaxos [OSDI ‘16], SpecPaxos [NSDI ’15] …

Neither approach is ideal: **reliable or performant**
The Two Different Worlds of Replication

How and where to store system state?

**World-1**

- Disk-durable
  - synchronously persist updates to disks
  - Paxos, Raft [ATC ‘14], ZAB [DSN ‘11], Gaios [NSDI ‘11], ZooKeeper, etcd, LogCabin …
  - safe but suffer from poor performance

**World-2**

- Memory-durable
  - buffer updates only in volatile memory
  - Viewstamped replication, NOPaxos [OSDI ‘16], SpecPaxos [NSDI ‘15] …

Neither approach is ideal: **reliable or performant**
The Two Different Worlds of Replication

How and where to store system state?

World-1

Disk-durable

- synchronously persist updates to disks

- safe but suffer from poor performance

- Paxos, Raft [ATC ‘14], ZAB [DSN ‘11], Gaios [NSDI ‘11], ZooKeeper, etcd, LogCabin …

World-2

Memory-durable

- buffer updates only in volatile memory

- performant but risk unsafety or unavailability

- Viewstamped replication, NOPaxos [OSDI ‘16], SpecPaxos [NSDI ‘15] …

Neither approach is ideal: reliable or performant
Can a protocol provide **strong reliability** while maintaining **high performance**?
SAUCR:
Situation-Aware Updates and Crash Recovery
SAUCR: Situation-Aware Updates and Crash Recovery

Simple insight: dynamically (based on the situation) decide how to commit updates
SAUCR: Situation-Aware Updates and Crash Recovery

Simple insight: dynamically (based on the situation) decide how to commit updates

→ with many or all nodes up, buffer in memory – fast mode
→ with failures, if only bare majority up, flush to disk – slow mode
SAUCR: Situation-Aware Updates and Crash Recovery

Simple insight: dynamically (based on the situation) decide how to commit updates
- with many or all nodes up, buffer in memory – fast mode
- with failures, if only bare majority up, flush to disk – slow mode

Strong reliability while maintaining high performance
Simultaneity of Failures

SAUCR’s effectiveness depends upon **simultaneity** of failures
Simultaneity of Failures

SAUCR’s effectiveness depends upon **simultaneity** of failures

- independent and non-simultaneous correlated (gap of a few milliseconds to a few seconds)
  - can react and switch from fast to slow mode
  - preserves durability and availability
Simultaneity of Failures

SAUCR’s effectiveness depends upon simultaneity of failures

- independent and non-simultaneous correlated (gap of a few milliseconds to a few seconds)
  - can react and switch from fast to slow mode
  - preserves durability and availability

- many truly simultaneous correlated
  - no gap and so cannot react
  - remain unavailable
Simultaneity of Failures

SAUCR’s effectiveness depends upon **simultaneity** of failures

- independent and non-simultaneous correlated (gap of a few milliseconds to a few seconds)
  - can react and switch from fast to slow mode
  - preserves durability and availability

- many **truly simultaneous correlated**
  - no gap and so cannot react
  - remain unavailable

- however, existing data hints they are extremely rare – the Non-Simultaneity Conjecture
Results
Results

Implemented in ZooKeeper
Results

Implemented in ZooKeeper

SAUCR **improves reliability** compared to memory-durable systems

- durable and available in 100s of crash scenarios
- memory-durable loses data or becomes unavailable
Results

Implemented in ZooKeeper

SAUCR improves reliability compared to memory-durable systems
- durable and available in 100s of crash scenarios
- memory-durable loses data or becomes unavailable

Improvements at no or little cost
- overheads within 0%-9% of memory-durable systems
Results

Implemented in ZooKeeper

SAUCR **improves reliability** compared to memory-durable systems
- **durable and available in 100s of crash scenarios**
- **memory-durable loses data or becomes unavailable**

**Improvements at no or little cost**
- **overheads within 0%-9% of memory-durable systems**

**Compared to disk-durable**
- **slight reduction in availability in extremely rare cases**
- **improves performance – 2.5x on SSDs, 100x on HDDs**
Outline

Introduction

Distributed updates and crash recovery
  - disk-durable protocols
  - memory-durable protocols

Situation-aware updates and crash recovery

Results

Summary and conclusion
Disk-Durable Protocols
Disk-Durable Protocols

Update

Leader

Follower

Follower
Disk-Durable Protocols

Update
Disk-Durable Protocols

Update

Client
A=2

Leader
A=1

Follower
A=1

Follower
A=1
Disk-Durable Protocols

Update

Client

Leader

Follower

Follower
Disk-Durable Protocols

Update

Client

Leader

Follower

Follower
Disk-Durable Protocols

Update

Client

Leader

Follower

Follower

A=1 | A=2

A=1 | A=2

A=1 | A=2

fsync

fsync

fsync
Disk-Durable Protocols

Update

Committed

Client

fsync completed on a majority?

Leader

fsync

A=1 A=2

Follower

fsync

A=1 A=2

Follower

fsync

A=1 A=2

A=1

A=2
Disk-Durable Protocols

Update

Committed

Client

fsync completed on a majority?

Leader

A=1  A=2

Follower

A=1  A=2

Follower

A=1  A=2

Recovery
Disk-Durable Protocols

Update

Committed

Client

fsync completed on a majority?

 Leader

 A=1 | A=2

Follower

 A=1 | A=2

Follower

 A=1 | A=2

Recovery

if ack’ed anyone, data on disk – safe
Disk-Durable Protocols

Update
Committed
Client

fsync completed on a majority?

Leader
Follower
Follower

Recovery
if ack’d anyone, data on disk – safe

A=1 A=2
A=1 A=2
A=1 A=2

fsync
fsync
fsync
Disk-Durable Protocols

Update

1. fsync completed on a majority?
2. If ack'd anyone, data on disk – safe

Recovery

Committed

Client

Leader

Follower

Follower

A=1  A=2

A=1  A=2

A=1  A=2
Disk-Durable Protocols

Update

Committed

Client

fsync completed on a majority?

Leader

Follower

Follower

fsync

fsync

fsync

A=1 A=2

A=1 A=2

A=1 A=2

Recovery

if ack’d anyone, data on disk – safe

A=1 A=2

OSDI’18
Disk-Durable Protocols

**Update**
- Client
- Committed
- fsync completed on a majority?

**Recovery**
- if ack’d anyone, data on disk – safe
- recovery: just read from local disk
  - ready
  - A=1 | A=2
  - immediate
Disk-Durable Protocols

### Update

- **Client** initiates an update.
- **Leader** synchronizes with the **Follower**.
- Fsync is completed on a majority.

### Recovery

- Fsync completed on a majority?
- If acknowledged by anyone, data on disk is safe.
- Recovery: just read from local disk.
- Immediate: ready
- Lagging: immediate

OSDI'18
Disk-Durable Protocols

Update

Committed

Client

fsync completed on a majority?

Leader

A=1  A=2

Follower

A=1  A=2

Follower

A=1  A=2

Recovery

if ack’d anyone, data on disk – safe

ready

immediate

recovery: just read from local disk

lagging

immediate

ready

Safe and available
Disk-Durable Protocols

Update

Committed

Client

fsync completed on a majority?

Leader

A=1

A=2

Follower

A=1

A=2

Follower

A=1

A=2

Recovery

if ack’d anyone, data on disk – safe

ready

immediate

A=1

A=2

recovery: just read from local disk

lagging

immediate

A=1

ready

Safe and available

But poor performance due to fsync – 50x on HDDs, 2.5x on SSDs
Memory-Durable Protocols (Oblivious Recovery)

Update

Client
A=2

Leader
-memory
A=1

Follower
-memory
A=1

Follower
-memory
A=1
Memory-Durable Protocols (Oblivious Recovery)

Update

buffered on a majority?

Committed

Client

Leader

Memory

A=1 A=2

Follower

Memory

A=1 A=2

Follower

Memory

A=1 A=2
Memory-Durable Protocols (Oblivious Recovery)

Update

Committed

Client

buffered on a majority?

Leader

Memory

A=1  A=2

Follower

Memory

A=1  A=2

Follower

Memory

A=1  A=2

Recovery
Memory-Durable Protocols (Oblivious Recovery)

Update

Committed

Client

buffered on a majority?

Leader

Memory

A=1, A=2

Follower

Memory

A=1, A=2

Follower

Memory

A=1, A=2

Recovery

Oblivious: doesn’t realize loss on failure
Memory-Durable Protocols (Oblivious Recovery)

**Update**
- **Committed**
- **Client**
  - buffered on a majority?

**Recovery**
- **Oblivious**: doesn’t realize loss on failure

---

**Leader**
- Memory
  - A=1
  - A=2

**Follower**
- Memory
  - A=1
  - A=2

---

**Memory**
- A=1
- A=2
Memory-Durable Protocols (Oblivious Recovery)

Update

Committed

Client

buffered on a majority?

Memory
A=1 A=2

Leader

Memory
A=1 A=2

Follower

Memory
A=1 A=2

Follower

Recovery

Oblivious: doesn’t realize loss on failure

Memory
A=1 A=2
Memory-Durable Protocols (Oblivious Recovery)

**Update**

- **Leader**
  - Memory: A=1, A=2
- **Follower**
  - Memory: A=1, A=2
- **Follower**
  - Memory: A=1, A=2

- **Committed**
- **Client**

**Recovery**

- **Oblivious**: doesn’t realize loss on failure

- **Memory**

---

OSDI ‘18
Memory-Durable Protocols (Oblivious Recovery)

Update

Committed

Client

buffered on a majority?

Leader

Memory

A=1 A=2

Follower

Memory

A=1 A=2

Follower

Memory

A=1 A=2

Recovery

Oblivious: doesn’t realize loss on failure

Immediate

ready

Memory
Memory-Durable Protocols (Oblivious Recovery)

**Update**
- Committed
- Client
  - buffered on a majority?

**Recovery**
- **Oblivious**: doesn’t realize loss on failure
- **Ready**
  - immediate
  - Memory

- **Leader**
  - Memory
  - A=1, A=2

- **Followers**
  - Memory
  - A=1, A=2

E.g., ZooKeeper with `forceSync = false` practitioners do use this config!
Memory-Durable Protocols (Oblivious Recovery)

Update

- Committed
- Client
- buffered on a majority?

Memory
- Leader: A=1 A=2
- Follower: A=1 A=2
- Follower: A=1 A=2

Recovery

Oblivious: doesn’t realize loss on failure

Immediate

Memory

- Memory-Durable Protocols (Oblivious Recovery)
- Performant
- e.g., ZooKeeper with $\text{forceSync} = \text{false}$ practitioners do use this config!
Memory-Durable Protocols (Oblivious Recovery)

Update

- Committed
- Client
- buffered on a majority?

Leader
- Memory
  - A=1
  - A=2

Follower
- Memory
  - A=1
  - A=2

Follower
- Memory
  - A=1
  - A=2

Recovery

Oblivious: doesn’t realize loss on failure

- immediate
- ready

Memory

Performant

But can lead to data loss

e.g., ZooKeeper with `forceSync = false` practitioners do use this config!
Data Loss Example in Oblivious Approach
Data Loss Example in Oblivious Approach
Data Loss Example in Oblivious Approach

A=1
A=1
A=1
A=1
A=1 committed
Data Loss Example in Oblivious Approach

A=1 committed
two nodes slow or failed
Data Loss Example in Oblivious Approach

A=1 committed
two nodes slow or failed
Data Loss Example in Oblivious Approach

A=1 committed
two nodes slow or failed

A=1 crashes
Data Loss Example in Oblivious Approach

A=1 committed

two nodes slow or failed

A=1

A=1

A=1

crashes, recovers

A=1

A=1

A=1

A=1
Data Loss Example in Oblivious Approach

A=1 committed two nodes slow or failed

A=1
A=1
A=1

A=1
A=1
A=1

crashes, recovers loses its data but oblivious: immediately joins
Data Loss Example in Oblivious Approach

A=1 committed two nodes slow or failed

crashes, recovers loses its data but oblivious: immediately joins
Data Loss Example in Oblivious Approach

A=1 committed two nodes slow or failed

A=1 crashes, recovers loses its data but oblivious: immediately joins
Data Loss Example in Oblivious Approach

A=1 committed
two nodes slow or failed

A=1 crashes, recovers
loses its data
but oblivious:
immediately joins

lagging nodes along with recovered
node form majority;
lose committed update

majority do not
know of previously
committed update
Memory-Durable Protocols (Loss-Aware Recovery)

Update

Committed

buffered on a majority?

Client

Memory
A=1 A=2
Leader

Memory
A=1 A=2
Follower

Memory
A=1 A=2
Follower
Memory-Durable Protocols (Loss-Aware Recovery)

Update

Committed

Client

buffered on a majority?

Leader

Follower

Follower

Memory
A=1 A=2

Memory
A=1 A=2

Memory
A=1 A=2

Recovery
Memory-Durable Protocols (Loss-Aware Recovery)

**Update**
- Client
- Committed
- buffered on a majority?

**Recovery**
- Loss-aware: realizes loss, waits for majority

- **Leader**
  - Memory
  - A=1, A=2

- **Followers**
  - Memory
  - A=1, A=2
Memory-Durable Protocols (Loss-Aware Recovery)

**Update**

- Committed
- Client
- buffered on a majority?

**Recovery**

- **Loss-aware**: realizes loss, waits for majority

```
Memory
A=1 A=2
Leader

Memory
A=1 A=2
Follower

Memory
A=1 A=2
Follower
```

Memory-Durable Protocols (Loss-Aware Recovery)
Memory-Durable Protocols (Loss-Aware Recovery)

**Update**

- Committed
- Client
- buffered on a majority?
- Memory
  - A=1
  - A=2
- Leader
- Memory
  - A=1
  - A=2
- Follower
- Memory
  - A=1
  - A=2
- Follower
- Memory
  - A=1
  - A=2
- Follower

**Recovery**

- Loss-aware: realizes loss, waits for majority
- Memory
  - A=1
  - A=2

OSDI '18
Memory-Durable Protocols (Loss-Aware Recovery)

Update

Committed

Client

buffered on a majority?

Memory
A=1  A=2
Leader

Memory
A=1  A=2
Follower

Memory
A=1  A=2
Follower

Recovery

Loss-aware: realizes loss, waits for majority

Memory

OSDI '18

OSDI '18
Memory-Durable Protocols (Loss-Aware Recovery)

**Update**

- **Committed**
- **Client**

```
buffered on a majority?
```

**Leader**

```
Memory
A=1  A=2
```

**Follower**

```
Memory
A=1  A=2
```

**Follower**

```
Memory
A=1  A=2
```

**Recovery**

- **Loss-aware**: realizes loss, waits for majority

```
recovering
wait for majority responses
```
Memory-Durable Protocols (Loss-Aware Recovery)

Update

Committed

Client

buffered on a majority?

Memory
A=1    A=2
Leader

Memory
A=1    A=2
Follower

Memory
A=1    A=2
Follower

Recovery

Loss-aware: realizes loss, waits for majority

recovering wait for majority responses

ready

Memory
A=1    A=2
Memory-Durable Protocols (Loss-Aware Recovery)

**Update**
- Leader
  - Memory: A=1, A=2
- Follower
  - Memory: A=1, A=2

Committed

**Recovery**
- Loss-aware: realizes loss, waits for majority
- Recovering
  - Wait for majority responses
  - Majority responses
  - Memory: A=1, A=2

*Ready*

E.g., Viewstamped replication

OSDI '18
Memory-Durable Protocols (Loss-Aware Recovery)

**Update**
- Committed
- Client
- buffered on a majority?

**Recovery**
- Loss-aware: realizes loss, waits for majority
- recovering
- wait for majority responses
- ready
- Memory
  - A=1
  - A=2

E.g., Viewstamped replication

Avoids loss (unlike oblivious) but can lead to unavailable
Unavailability Example in Loss-Aware Approach
Unavailability Example in Loss-Aware Approach

A=1 committed
two nodes crashed
Unavailability Example in Loss-Aware Approach

A=1 committed
two nodes crashed
Unavailability Example in Loss-Aware Approach

A=1 committed

two nodes crashed

A=1 crashes
Unavailability Example in Loss-Aware Approach

A=1 committed

two nodes crashed

crashes, recovers
Unavailability Example in Loss-Aware Approach

A=1 committed
A=1
A=1
A=1

A=1 crashed, recovers
A=1
A=1

Although majority up – unavailable
Unavailability Example in Loss-Aware Approach

A=1 committed two nodes crashed

A=1 crashes, recovers cannot collect majority responses although majority up – unavailable
Unavailability Example in Loss-Aware Approach

A=1 committed

two nodes crashed

A=1

A=1

A=1

A=1

A=1

crashes, recovers

cannot collect

majority responses

although majority up – unavailable

failed nodes recover

A=1

A=1

A=1

A=1

A=1

A=1

A=1
Unavailability Example in Loss-Aware Approach

A=1 committed
two nodes crashed

A=1
A=1
A=1

A=1 crashes, recovers
cannot collect
majority responses
although majority up – unavailable

A=1 failed nodes recover
stay in recovering
unavailable even after
all nodes recover
Outline

Introduction

Distributed updates and crash recovery

Situation-aware updates and crash recovery
- SAUCR insights, guarantees, and overview
- situation-aware updates
- situation-aware crash recovery

Results

Summary and conclusion
SAUCR Intuition and Insight
SAUCR Intuition and Insight

Existing protocols are static in nature: do not adapt to failures
SAUCR Intuition and Insight

Existing protocols are **static in nature**: do not adapt to failures

- always
- **Memory-durable**
- buffer even with many failures
- poor reliability
SAUCR Intuition and Insight

Existing protocols are static in nature: do not adapt to failures

- **Memory-durable**
  - always buffer even with many failures
  - poor reliability

- **Disk-durable**
  - always persist even when no failures
  - poor performance
SAUCR Intuition and Insight

Existing protocols are *static in nature*: do not adapt to failures

- **Memory-durable**
  - buffer even with many failures
  - poor reliability

- **Disk-durable**
  - persist even when no failures
  - poor performance

**Insight:** reacting to failures and adapting to situation can achieve reliability and performance
SAUCR Intuition and Insight

Existing protocols are **static in nature**: do not adapt to failures

- **Memory-durable**: buffer even with many failures, poor reliability
- **Disk-durable**: persist even when no failures, poor performance
- **Memory-durable**: buffer in memory

**Insight**: reacting to failures and adapting to situation can achieve reliability and performance

- when no or few failures could buffer in memory
SAUCR Intuition and Insight

Existing protocols are **static in nature**: do not adapt to failures

- **Memory-durable**: always
  - buffer even with many failures
  - poor reliability
- **Disk-durable**: always
  - persist even when no failures
  - poor performance
- **Memory-durable**: common case
  - when many or all up
  - buffer in memory
- **Disk-durable**: with failures
  - when only minimum up
  - flush to disk

**Insight**: reacting to failures and adapting to situation can achieve reliability and performance

- when no or few failures could buffer in memory
- when failure arise, flush
Guarantees Depend upon Simultaneity of Failures
Guarantees Depend upon **Simultaneity** of Failures

With *non-simultaneous*, gap exists, SAUCR can react and ensures durability
Guarantees Depend upon **Simultaneity** of Failures

With **non-simultaneous**, gap exists, SAUCR can react and ensures durability

→ **independent**: likelihood of many nodes failing together is negligible
Guarantees Depend upon **Simultaneity** of Failures

With **non-simultaneous**, gap exists, SAUCR can react and ensures durability

- **independent**: likelihood of many nodes failing together is negligible
- **correlated**: many nodes fail together
  - although many nodes fail, *not necessarily simultaneous*; most cases, non-simultaneous
Guarantees Depend upon **Simultaneity** of Failures

With **non-simultaneous**, gap exists, SAUCR can react and ensures durability

- **independent**: likelihood of many nodes failing together is negligible
- **correlated**: many nodes fail together
  - although many nodes fail, not necessarily **simultaneous**; most cases, non-simultaneous

With **simultaneous correlated**, no gap, SAUCR cannot react, unavailable
Guarantees Depend upon Simultaneity of Failures

With **non-simultaneous**, gap exists, SAUCR can react and ensures durability

- **independent**: likelihood of many nodes failing together is negligible
- **correlated**: many nodes fail together
  - although many nodes fail, *not necessarily simultaneous*; most cases, non-simultaneous

With **simultaneous correlated**, no gap, SAUCR cannot react, unavailable

We conjecture they are extremely rare: a gap exists between failures

- correlated but a few seconds apart [Ford et al., OSDI ‘10]
- analysis reveals a gap of 50 ms or more almost always
Guarantees Depend upon **Simultaneity** of Failures

With **non-simultaneous**, gap exists, SAUCR can react and ensures durability
- independent: likelihood of many nodes failing together is negligible
- correlated: many nodes fail together
  - although many nodes fail, *not necessarily simultaneous*; most cases, non-simultaneous

With **simultaneous correlated**, no gap, SAUCR cannot react, unavailable

We conjecture they are extremely rare: a gap exists between failures
- correlated but a few seconds apart [Ford et al., OSDI ‘10]
- analysis reveals a gap of 50 ms or more almost always

**Most cases**: any no. of independent and non-simultaneous correlated – same as disk-durable
**Rare cases**: more than a majority crash truly simultaneously – remain unavailable
SAUCR Overview
SAUCR Overview

Updates

→ when more than a majority up, buffer updates in memory – fast mode
→ e.g., 4 or 5 nodes up in a 5-node cluster
SAUCR Overview

Updates

- when more than a majority up, buffer updates in memory – fast mode
  - e.g., 4 or 5 nodes up in a 5-node cluster
- When nodes fail and only a bare majority alive, flush to disk – slow mode
  - e.g., only 3 nodes up in a 5-node cluster
SAUCR Overview

Updates
- when more than a majority up, buffer updates in memory – fast mode
  - e.g., 4 or 5 nodes up in a 5-node cluster
- When nodes fail and only a bare majority alive, flush to disk – slow mode
  - e.g., only 3 nodes up in a 5-node cluster

Crash Recovery
- when a node recovers from a crash, it recovers its data
  - either from its disk (if crashed in slow mode)
  - or from other nodes (if crashed in fast mode)
Situation-Aware Updates
Situation-Aware Updates

all nodes up
Situation-Aware Updates

all nodes up
fast mode -
buffer updates
Situation-Aware Updates

all nodes up

fast mode - buffer updates

4 nodes up

(more than majority)
Situation-Aware Updates

*all nodes up*

*fast mode - buffer updates*

*4 nodes up (more than majority) remain in fast mode*
Situation-Aware Updates

- All nodes up
  - Fast mode - buffer updates

- 4 nodes up (more than majority)
  - Buffer updates remain in fast mode

- Only majority up
Situation-Aware Updates

- **all nodes up**
  - Fast mode - buffer updates

- **4 nodes up** (more than majority)
  - Remain in fast mode

- **only majority up**
  - Switch to slow, flush to disk
Situation-Aware Updates

- All nodes up
  - Fast mode - buffer updates

- 4 nodes up (more than majority)
  - Remain in fast mode

- Only majority up
  - Switch to slow, flush to disk
Situation-Aware Updates

- **all nodes up**: fast mode - buffer updates
- **4 nodes up** (more than majority): remain in fast mode
- **only majority up**: switch to slow, flush to disk
- **commit**: subsequent updates in slow mode
Situation-Aware Updates

- All nodes up
  - Fast mode - buffer updates

- 4 nodes up (more than majority)
  - Remain in fast mode

- Only majority up
  - Switch to slow, flush to disk

- Commit subsequent updates in slow mode

- One node recovers and catches up;
Situación-Aware Updates

- **All nodes up**: fast mode - buffer updates
- **4 nodes up (more than majority)**: remain in fast mode
- **Only majority up**: switch to slow, flush to disk
- **Commit subsequent updates in slow mode**: one node recovers and catches up;
Situation-Aware Updates

all nodes up
fast mode - buffer updates

4 nodes up (more than majority) remain in fast mode

only majority up
switch to slow, flush to disk

commit subsequent updates in slow mode

one node recovers and catches up;
Situation-Aware Updates

- **all nodes up**
  - fast mode - buffer updates

- **4 nodes up**
  - (more than majority) remain in fast mode

- **only majority up**
  - switch to slow, flush to disk

- **commit**
  - subsequent updates in slow mode

- **one node recovers**
  - and catches up; switch to fast
Failure Reaction

Basic failure-detection mechanism: heartbeats

Follower failures

Leader failures

Challenges: too many packets, spurious elections, too much data to flush

Techniques in the paper …

Result: can react to failures even when they are only a few milliseconds apart, preserving durability and availability
Mode-Aware Crash Recovery
Mode-Aware Crash Recovery

Disk-durable: *always* recover from *disk*
Memory-durable: *always* recover from *other nodes* (loss-aware)
Mode-Aware Crash Recovery

Disk-durable: always recover from disk
Memory-durable: always recover from other nodes (loss-aware)
Mode-Aware Crash Recovery

Disk-durable: always recover from disk
Memory-durable: always recover from other nodes (loss-aware)
Mode-Aware Crash Recovery

Disk-durable: *always* recover from *disk*
Memory-durable: *always* recover from *other nodes* (loss-aware)

SAUCR
Mode-Aware Crash Recovery

Disk-durable: always recover from disk
Memory-durable: always recover from other nodes (loss-aware)
Mode-Aware Crash Recovery

Disk-durable: *always* recover from *disk*

Memory-durable: *always* recover from *other nodes* (loss-aware)
Mode-Aware Crash Recovery

Disk-durable: *always* recover from *disk*
Memory-durable: *always* recover from *other nodes* (loss-aware)
Mode-Aware Crash Recovery

Disk-durable: *always* recover from *disk*
Memory-durable: *always* recover from *other nodes* (loss-aware)
Mode-Aware Crash Recovery

Disk-durable: always recover from disk
Memory-durable: always recover from other nodes (loss-aware)
Mode-Aware Crash Recovery

Disk-durable: always recover from disk
Memory-durable: always recover from other nodes (loss-aware)
Mode-Aware Crash Recovery

Disk-durable: always recover from disk
Memory-durable: always recover from other nodes (loss-aware)
Intuition for why SAUCR’s recovery is safe
Intuition for why SAUCR’s recovery is safe

Assume *update-A* committed, S1 recovers and has seen *A* before crash.
Intuition for why SAUCR’s recovery is safe

Assume update-A committed, S1 recovers and has seen A before crash

Safety condition: update-A must be recovered
Intuition for why SAUCR’s recovery is safe

Assume update-A committed, S1 recovers and has seen A before crash

Safety condition: update-A must be recovered
If A was committed in fast mode, then at least one in any bare minority must contain update-A
Intuition for why SAUCR’s recovery is safe

Assume update-A committed, SI recovers and has seen A before crash

Safety condition: update-A must be recovered
If A was committed in fast mode, then at least one in any bare minority must contain update-A
If update-A was committed in slow mode, SI recovers from disk
Intuition for why SAUCR’s recovery is safe

Assume update-A committed, S1 recovers and has seen A before crash

Safety condition: update-A must be recovered
If A was committed in fast mode, then at least one in any bare minority must contain update-A
If update-A was committed in slow mode, S1 recovers from disk
Proof sketch in the paper …
Outline

Introduction
Distributed updates and crash recovery
Situation-aware updates and crash recovery
Results
Summary and conclusion
Evaluation

We implement in SAUCR in ZooKeeper

Compare SAUCR’s reliability and performance against
- disk-durable ZooKeeper (forceSync = true)
- memory-durable ZooKeeper (forceSync = false)
- viewstamped replication (ideal model)
Reliability Testing

Cluster crash-testing framework
Generates cluster-state sequences

How it works?
Please see our paper…
## Reliability Results

<table>
<thead>
<tr>
<th>Systems</th>
<th>Non-Simultaneous</th>
<th>Simultaneous</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Correct</td>
<td>Unavailable</td>
</tr>
<tr>
<td>memory-durable zookeeper</td>
<td>703</td>
<td>0</td>
</tr>
<tr>
<td>viewstamped replication</td>
<td>217</td>
<td>1047</td>
</tr>
<tr>
<td>disk-durable zookeeper</td>
<td>1264</td>
<td>0</td>
</tr>
<tr>
<td>SAUCR</td>
<td>1264</td>
<td>0</td>
</tr>
</tbody>
</table>

- Non-simultaneous: gap of 50 ms, simultaneous: no gap
- Memory-durable zookeeper *silently loses data*
- Viewstamped replication leads to *permanent unavailability*
- SAUCR reacts to non-simultaneous – *durable and available*
- Other systems *behave the same* as non-simultaneous cases
- Simultaneous: SAUCR by design remains unavailable in some cases
Macro-benchmark Performance: YCSB-load

Compared to disk-durable, both memory-durable and SAUCR are faster. SAUCR’s performance matches memory-durable ZooKeeper within 9% of memory-durable Zookeeper even for write-intensive workloads. Overheads because SAUCR writes to one additional node.
Summary

Replication protocols are an important foundation need to be performant, yet also provide high reliability.

**Dichotomy**: disk-durable vs. memory-durable protocols
unsavory choices: either performant or reliable

**SAUCR** – situation-aware updates and crash recovery
provides both high performance and reliability
Conclusions

Paying careful attention to how failures occur
- can find approaches that provide both performance and reliability
- more data from real-world deployments?

Hybrid approach – an effective systems-design technique – applicable to distributed updates and recovery too
- worthwhile to look at other important protocols/systems where we make similar two-ends-of-the-spectrum tradeoffs?

Thank you!
Poster #6