www.storagedeveloper.org

Protocol-Aware Recovery for Consensus-based Storage

Ramnatthan Alagappan
University of Wisconsin – Madison

Failures in Distributed Storage Systems

System crashes

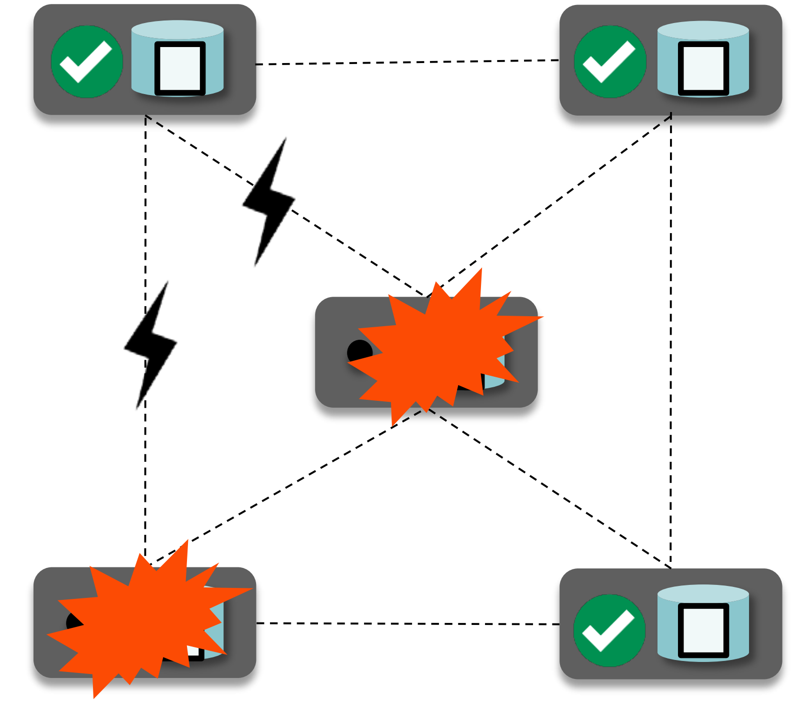
Network failures

redundancy masks failures

System as a whole unaffected

data is available

data is correct



2018 Storage Developer Conference. © University of Wisconsin - Madison. All Rights Reserved.

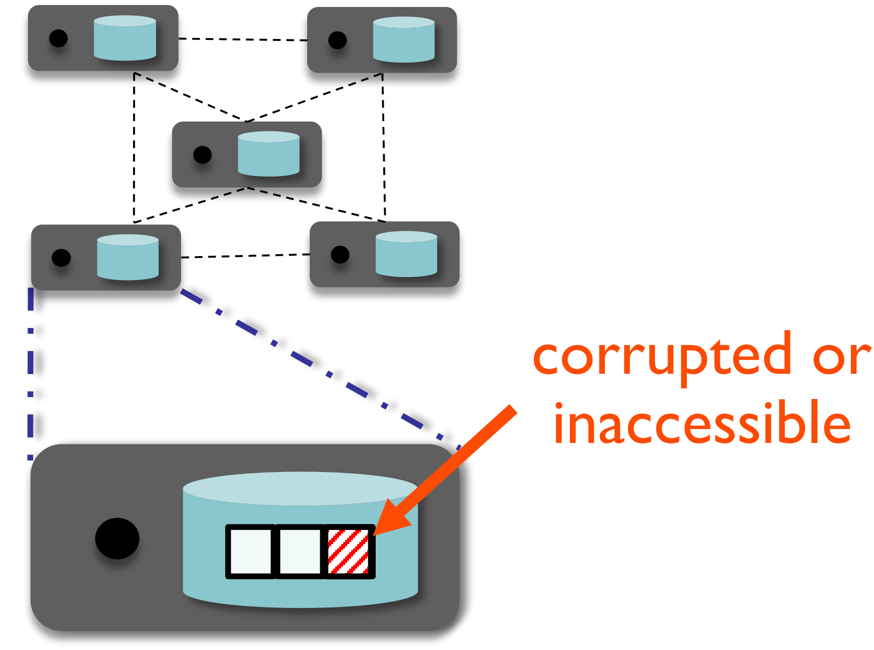
How About Faulty Data?

Data could be faulty

corrupted (disk corruption)

inaccessible (latent errors)

We call these storage faults



 © University of Wisconsin - Madison. All Rights Reserved.

Storage Corruptions and Errors Are Real

Latent errors in 8.5% of 1.5M drives [Bairavasundaram07]	400K checksum mismatches [Bairavasundaram08]	Flash Reliability [Schroeder16]	SSD Failures in Datacenters [Narayanan16]
Latent Sector Errors [Schroeder10]	Corruption Due to Misdirected Writes [Kruikov08]	Firmware bugs, media scratches etc., [Prabhakaran05]	Data Corruptions [Panzer-Steindel07]

Data corruption at massive scale

icant sample.
othing!

Testing copies=n resiliency

I decided to see how well ZFS copies=n would stand up to on-disk corruption today. Spoiler alert: not great.

ow

eople reacted with disbelief to my recent series on data corruption (see How data gets lost, 50 ways to ur data and How Microsoft puts your data at risk), claiming it had never happened to them. Really?

By Robin Harris for Storage Bits | September 17, 2007 -- 21:01 GMT (14:01 PDT) | Topic: Data Centers

2018 Storage Developer Conference. © University of Wisconsin - Madison. All Rights Reserved.

This talk…A “Measure-Then-Build” Approach

Part-1: Measure and understand how distributed systems react to storage faults

Part-2: Build a new recovery protocol that correctly recovers from storage faults (focus on RSM-based systems)

2018 Storage Developer Conference. © University of Wisconsin - Madison. All Rights Reserved.

Part-1: Measure

Behavior of eight systems in response to file-system faults

Main result: redundancy does **not** imply fault tolerance

a single fault in one node can cause catastrophic outcomes

	Silent corruption	Unavailability	Data loss	Reduced redundancy	Query failures
Redis	X	X	X	X	X
ZooKeeper		X	X	X	
Cassandra	X	X	X	X	
Kafka		X		X	X
RethinkDB	X			X	
MongoDB				X	
LogCabin				X	
CockroachDB		X	X	X	X

2018 Storage Developer Conference. © University of Wisconsin - Madison. All Rights Reserved.

Why does Redundancy Not Imply Fault Tolerance?

Some fundamental problems across systems – not just bugs!

Faults are often undetected locally – leads to harmful global effects

Crashing is the common action – redundancy underutilized

Crash and corruption handling are entangled – data loss

Unsafe interaction between local behavior and global distributed protocols can spread corruption or data loss

© University of Wisconsin - Madison. All Rights Reserved.

Part-2: Build

How to recover from storage faults?

Solve in an important class of systems: RSM

- based on Paxos, Raft (e.g., ZooKeeper, etcd)

CTRL (Corruption-Tolerant RepLication)

- safe and highly available with low performance overhead
- applied to LogCabin and ZooKeeper
- experimentally verified guarantees and little overheads (4%-8%)

2018 Storage Developer Conference. © University of Wisconsin - Madison. All Rights Reserved.

Outline

Introduction

Part-1: Measure

Part-2: Build

Summary

Conclusion

© University of Wisconsin - Madison. All Rights Reserved.

Fault Model

Client

Request

Server 1

Server 2

Server 3

read/ write

File System

Fault for current run:
server 1, block B1

Fault for next run:
server 1, block B1
read error

A single fault to a single file-system block in a single node

Faults injected only to user data not filesystem metadata

2018 Storage Developer Conference. © University of Wisconsin - Madison. All Rights Reserved.

Fault Model: ext4 and btrfs

Ext4: disk corruption → corrupted data to apps

Btrfs: disk corruption → I/O error to apps

2018 Storage Developer Conference. © University of Wisconsin - Madison. All Rights Reserved.

Fault Injection Methodology - Errfs

errfs - a FUSE file system to inject file-system faults

Client
Read
Server 1
Server 2
Server 3
read B1-B4
return B1'-B4
errfs (FUSE FS)
errfs (FUSE FS)
errfs (FUSE FS)
read B1-B4
return B1-B4

Fault for current run:
server 1, block B1

Local Behavior
Crash
Retry
Ignore faulty data
No detection/recovery

Global Effect
Corruption, Data loss, Unavailability

2018 Storage Developer Conference. © University of Wisconsin - Madison. All Rights Reserved.

Behavior Inference Methodology

Repeat for other blocks, other servers, other faults for different workloads

	Fault	Behavior Observed
Run 1	**Server**: server 1 **Block**: logical data structure X **Fault**: read corruption **Workload**: read	**Local Behavior**: Ignore faulty data **Global Effect**: Data loss
Run 2	**Server**: server 2 **Block**: logical data structure Y **Fault**: write error **Workload**: write	**Local Behavior**: Crash **Global Effect**: None
	⋮	⋮

2018 Storage Developer Conference. © University of Wisconsin - Madison. All Rights Reserved.

System Behavior Analysis

Behavior of eight distributed systems to file-system faults

Metadata stores: ZooKeeper, LogCabin

Wide column store: Cassandra

Document stores: MongoDB

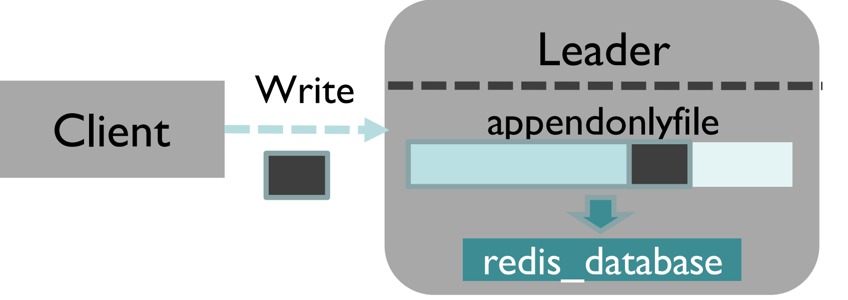
Distributed databases: RethinkDB, CockroachDB

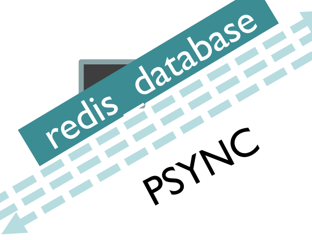
Message Queues: Kafka

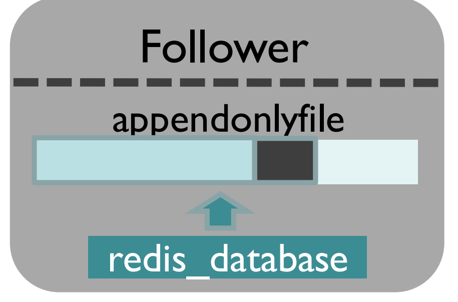
 © University of Wisconsin - Madison. All Rights Reserved.

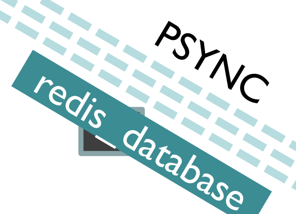
An Example: Redis

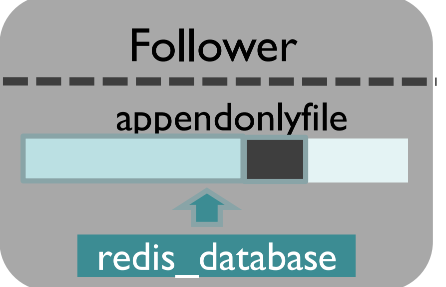
Redis is a popular data structure store











2018 Storage Developer Conference. © University of Wisconsin - Madison. All Rights Reserved.

Redis: Analysis

Read Workload

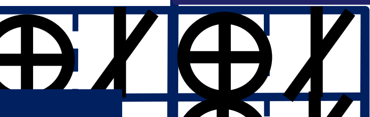
No checksums to detect corruption

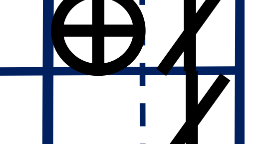
No checksums to detect corruption

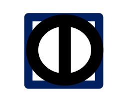
Leader returns corrupted data on queries

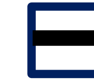
Corruption propagation to followers

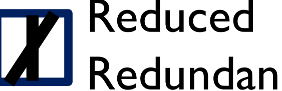
F Follower

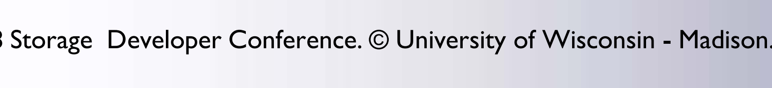












2018 Storage Developer Conference. © University of Wisconsin - Madison. All Rights Reserved.

Other Systems

Metadata stores: ZooKeeper, LogCabin

Wide column store: Cassandra

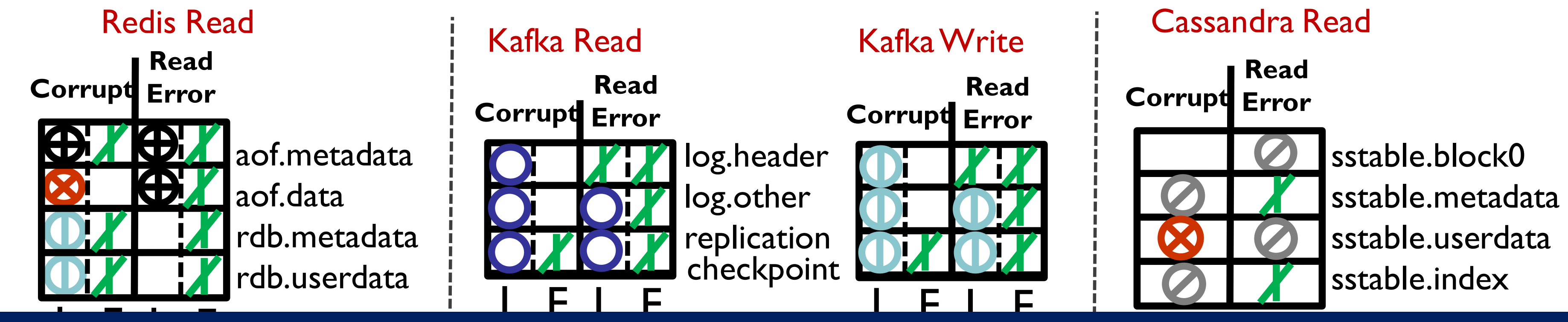
Document stores: MongoDB

Distributed databases: RethinkDB, CockroachDB

Message Queues: Kafka

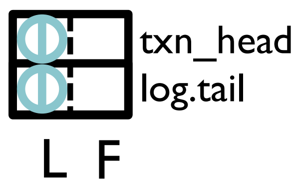
2018 Storage Developer Conference. © University of Wisconsin - Madison. All Rights Reserved.

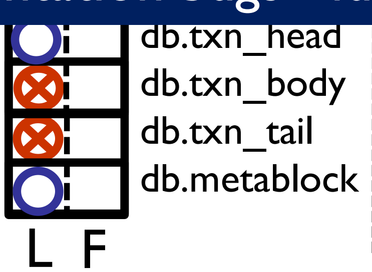
Redundancy Does not Provide Fault Tolerance



Harmful global effects despite redundancy

Not simple implementation bugs - fundamental problems across multiple systems!





Data Loss

Write Unavailability

Query Failure

Reduced Redundancy

2018 Storage Developer Conference. © University of Wisconsin - Madison. All Rights Reserved.

Why does Redundancy Not Imply Fault Tolerance?

Fundamental problems across systems – not just bugs!

Faults are often undetected locally – leads to harmful global effects

Crashing is the common action – redundancy underutilized

Crash and corruption handling are entangled – data loss

Unsafe interaction between local behavior and global distributed protocols can spread corruption or data loss

Why does Redundancy Not Imply Fault Tolerance?

Faults are often undetected locally – leads to harmful global effects

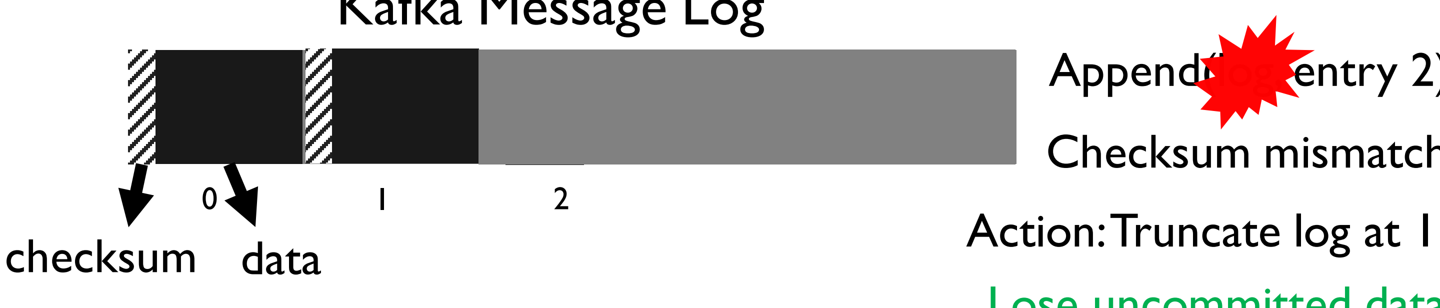
Crashing is the common action – redundancy underutilized

Crash and corruption handling are entangled – data loss

Unsafe interaction between local behavior and global distributed protocols can spread corruption or data loss

2018 Storage Developer Conference. © University of Wisconsin - Madison. All Rights Reserved.

Crash and Corruption Handling are Entangled



Need for discerning corruptions due to crashes from other type of corruptions

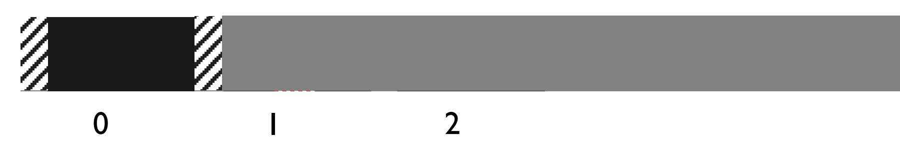
Developers of LogCabin and RethinkDB agree entanglement is the problem

2018 Storage Developer Conference. © University of Wisconsin - Madison. All Rights Reserved.

Unsafe Interaction between Local & Global Protocols

Kafka: Message log at Node 1

Local Behavior



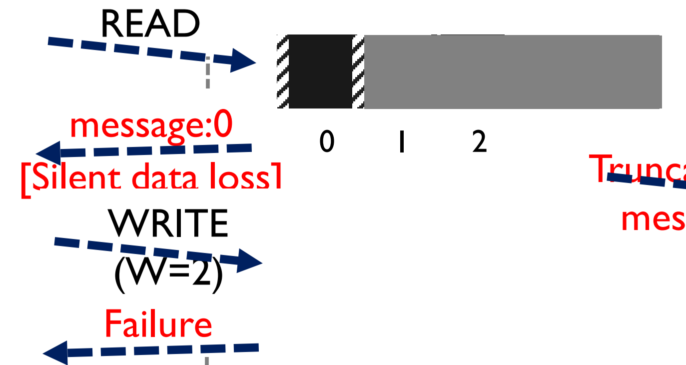
Disk corruption
Checksum mismatch
Action: Truncate log at 0
Lose committed data!

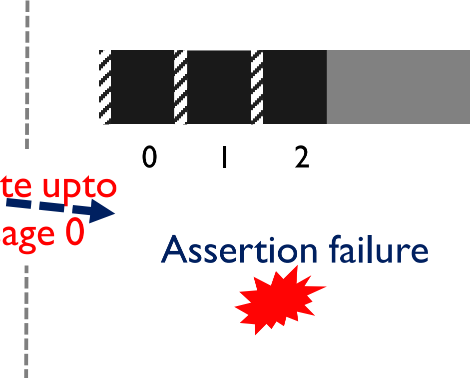
Need for synergy between local behavior and global protocol

Set of in-sync replicas

Node1 with truncated log not removed from in-sync replicas

Node 1 elected as leader

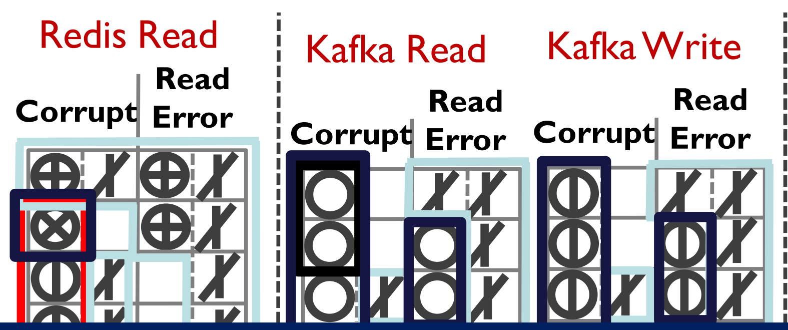




Unsafe interaction between local behavior and leader election protocol leads to data loss and write unavailability

2018 Storage Developer Conference. © University of Wisconsin - Madison. All Rights Reserved.

Why does Redundancy Not Imply Fault Tolerance?

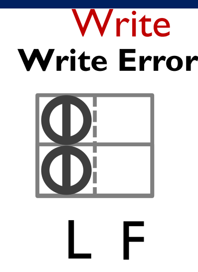


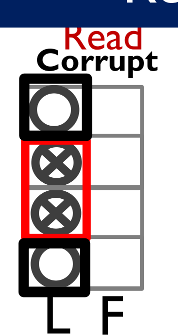
Faults are often locally undetected

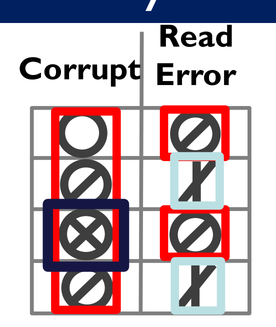
Crashing on detecting faults is the common reaction

Not simple implementation bugs - fundamental problems across multiple systems!
Redundancy underutilized as a source of recovery

entangled







Unsafe interaction between local and global protocols

2018 Storage Developer Conference. © University of Wisconsin - Madison. All Rights Reserved.

Part-1 Summary

We analyzed distributed storage reactions to single file-system faults

- Redis, ZooKeeper, Cassandra, Kafka, MongoDB, LogCabin, RethinkDB, and CockroachDB

Redundancy does not provide fault tolerance

- A single fault in one node can cause data loss, corruption, unavailability, and spread of corruption to other intact replicas

Some fundamental problems across multiple systems:

- Faults are often undetected locally – leads to harmful global effects
- On detection, crashing is the common action – redundancy underutilized
- Crash and corruption handling are entangled – loss of committed data
- Unsafe interaction between local behavior and global distributed protocols can spread corruption or data loss

© University of Wisconsin - Madison. All Rights Reserved.

Outline

Introduction

Part-1: Measure

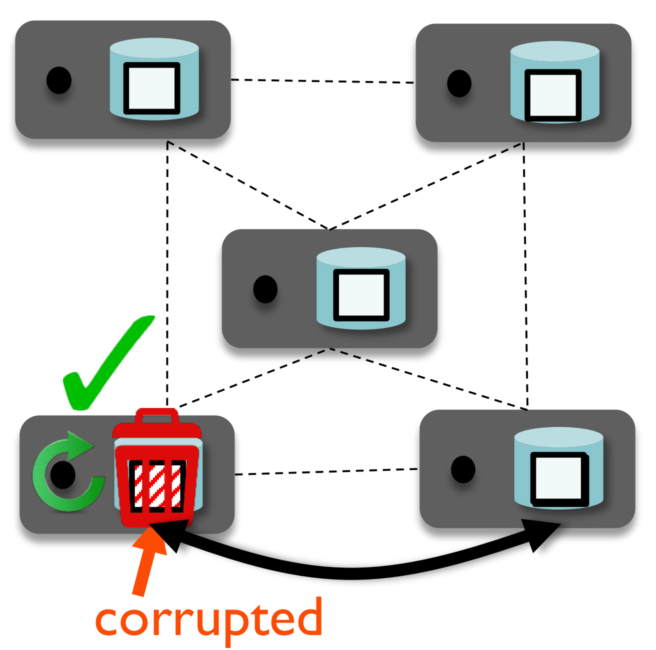
Part-2: Build

Summary

Conclusion

2018 Storage Developer Conference. © University of Wisconsin - Madison. All Rights Reserved.

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

The approach seems intuitive and works - all good, right?

> 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…

Looks reasonable: redundancy will help

2018 Storage Developer Conference. © University of Wisconsin - Madison. All Rights Reserved.

Unfortunately, No…Not So Easy!

Surprisingly, can lead to a global data loss!

SLOW

SLOW

This majority has no idea about the committed data

Committed data is lost!

2018 Storage Developer Conference. © University of Wisconsin - Madison. All Rights Reserved.

Problem: Approach is Protocol-Oblivious

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**

 © University of Wisconsin - Madison. All Rights Reserved.

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

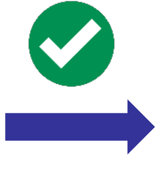
strong guarantees, even a small misstep can break

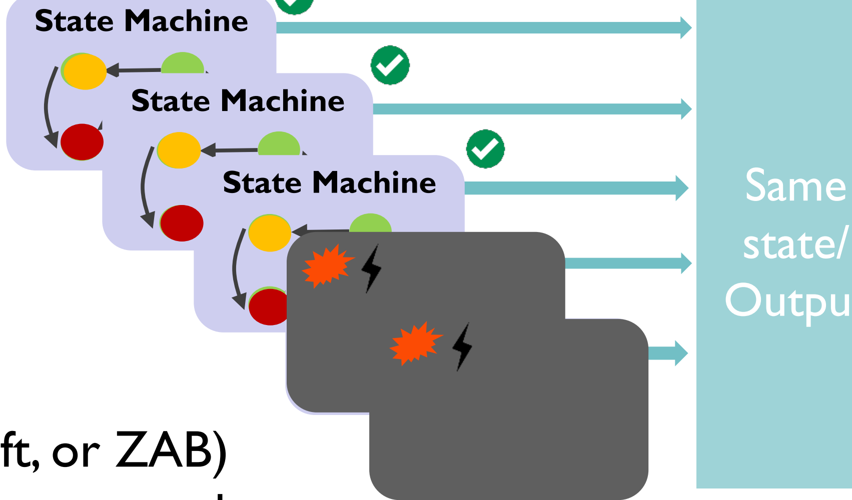
2018 Storage Developer Conference. © University of Wisconsin - Madison. All Rights Reserved.

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

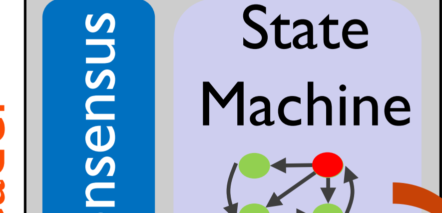
A consensus algorithm (e.g., Paxos, Raft, or ZAB) ensures SMs process commands in the same order

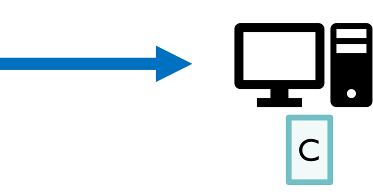
2018 Storage Developer Conference. © University of Wisconsin - Madison. All Rights Reserved.

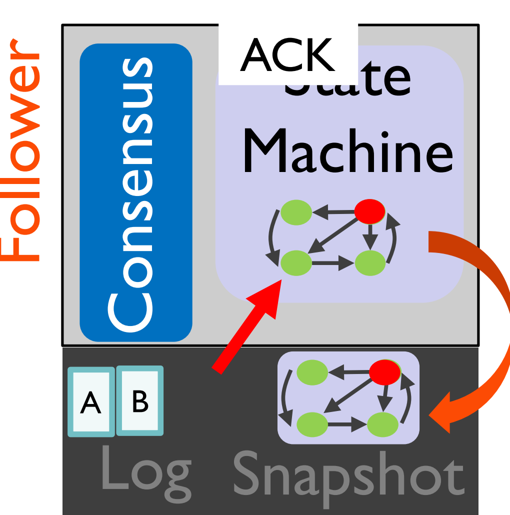
Replicated State Update

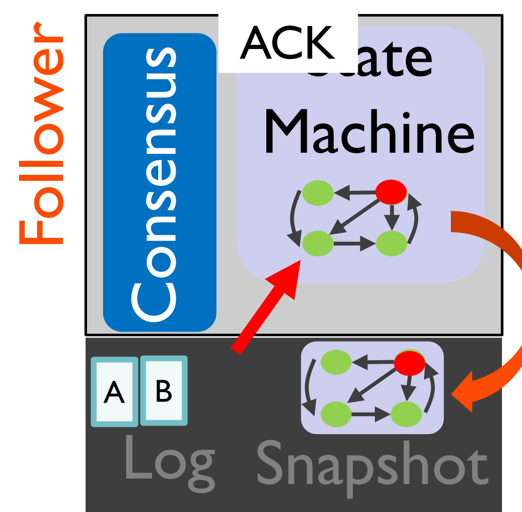
apply to SM once majority log the command

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









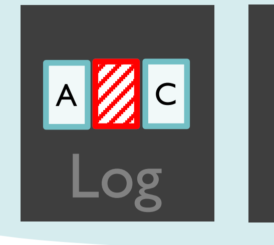
2018 Storage Developer Conference. © University of Wisconsin - Madison. All Rights Reserved.

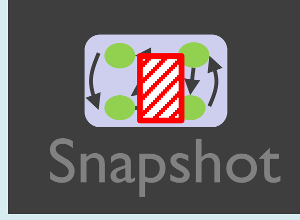
RSM Persistent Structures

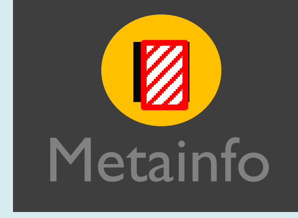
get corrupted data (e.g., ext2/3/4)
get error (e.g., any FS on latent errors, btrfs on a corruption)

read access

File System







disk corruption or latent sector errors

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

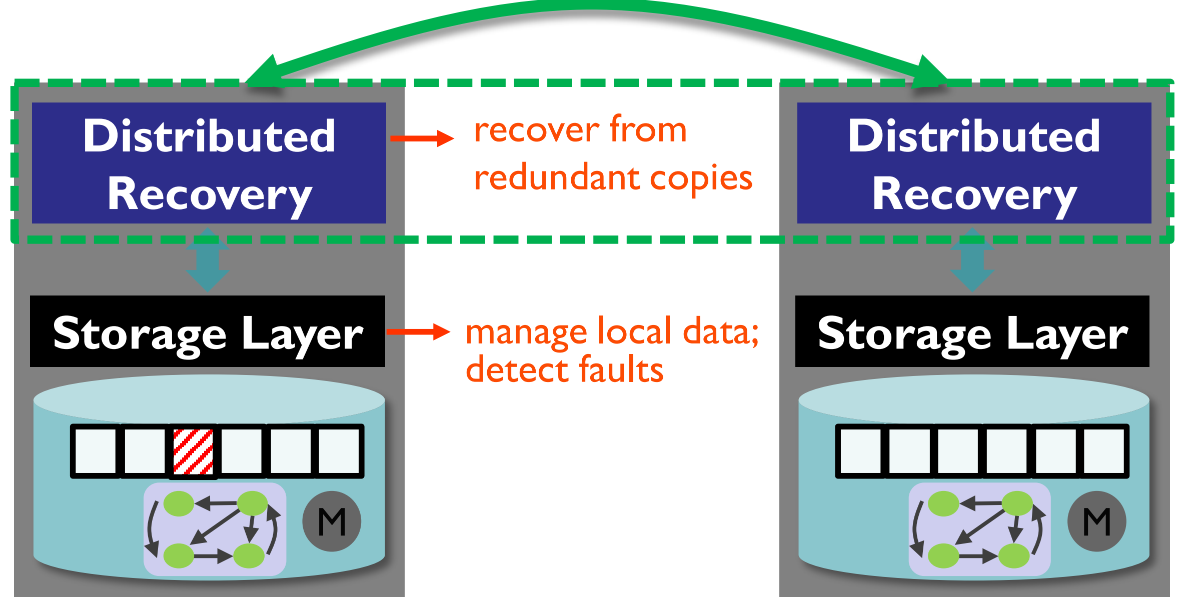
2018 Storage Developer Conference. © University of Wisconsin - Madison. All Rights Reserved.

CTRL Overview

Two components

Local storage layer

Distributed recovery



Exploit RSM knowledge to correctly and quickly recover faulty data

2018 Storage Developer Conference. © University of Wisconsin - Madison. All Rights Reserved.

CTRL Guarantees

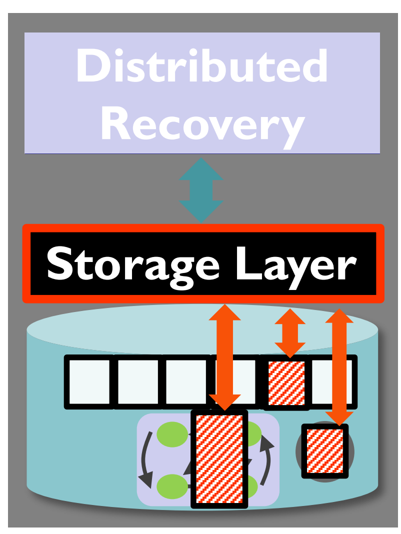
Committed data will never be lost

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

Provide the highest possible availability

2018 Storage Developer Conference. © University of Wisconsin - Madison. All Rights Reserved.

CTRL Local Storage



Main function: detect and identify

- whether log/snapshot/metainfo faulty or not?
- what is corrupted? (e.g., which log entry?)

Requirements

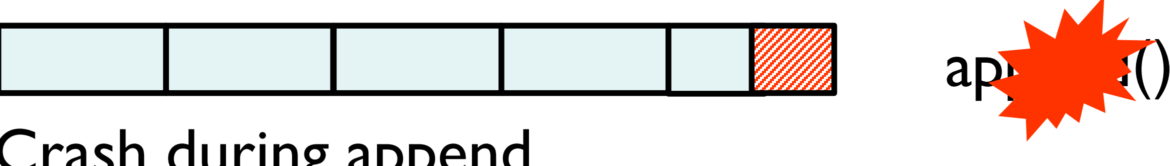
- low performance overheads
- low space overheads

An interesting problem: disentangling crashes and corruptions in log

- checksum mismatch due to crash or disk corruption?

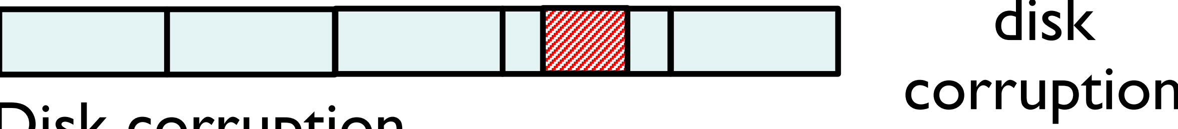
2018 Storage Developer Conference. © University of Wisconsin - Madison. All Rights Reserved.

Crash-Corruption Entanglement in the Log



Crash during append

- ➡ recovery: can truncate entry - unacknowledged



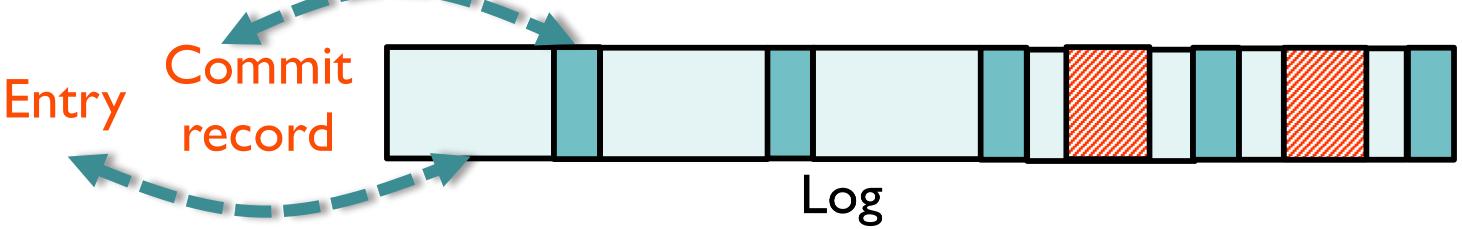
Disk corruption

- ➡ cannot truncate, may lose possibly committed data!

Current systems conflate the two conditions – always truncate

2018 Storage Developer Conference. © University of Wisconsin - Madison. All Rights Reserved.

Disentangling Crashes and Corruptions



If commit record not present, but checksum mismatch, then crashed in the middle of update – locally discard, skip recovery

If commit record present, but checksum mismatch, and a subsequent entry present, then a corruption

- however, if a subsequent entry is NOT present, then cannot determine whether corruption or crash

2018 Storage Developer Conference. © University of Wisconsin - Madison. All Rights Reserved.

Cannot Disentangle Last Entry Sometimes

last entry checksum mismatch, when commit record is present, could be either

Corruption

persisted safely
later corrupted

Crash

write(entry)
write(commit rec)
fsync(log)

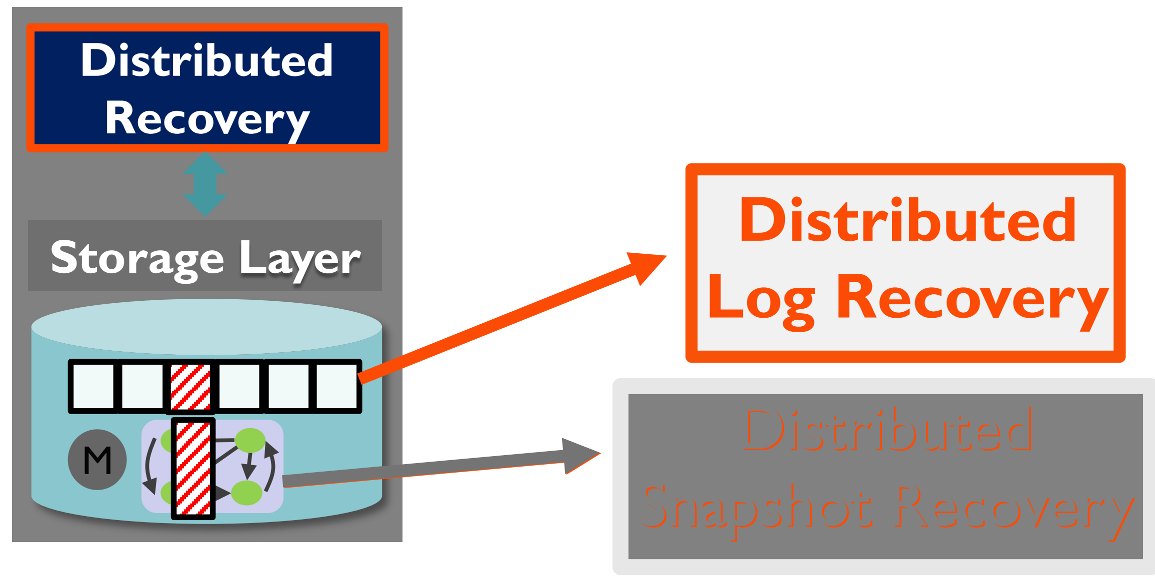
Fundamental limitation, not specific to CTRL

If cannot disentangle, safely mark as corrupted

➡ leave to distributed recovery to handle

2018 Storage Developer Conference. © University of Wisconsin - Madison. All Rights Reserved.

CTRL Distributed Recovery



2018 Storage Developer Conference. © University of Wisconsin - Madison. All Rights Reserved.

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

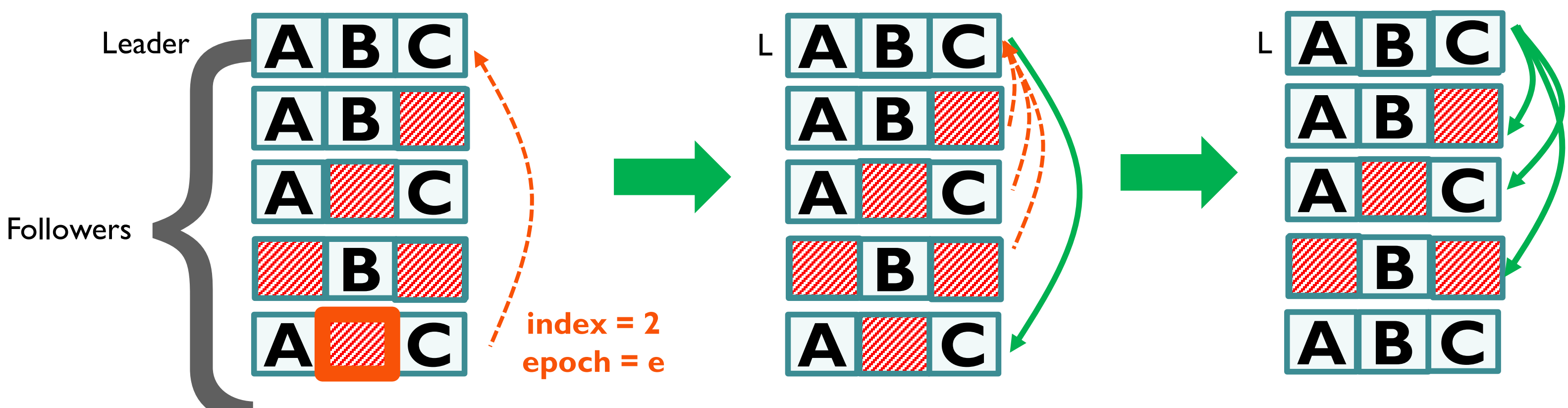
CTRL exploits these properties to perform recovery

2018 Storage Developer Conference. © University of Wisconsin - Madison. All Rights Reserved.

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!

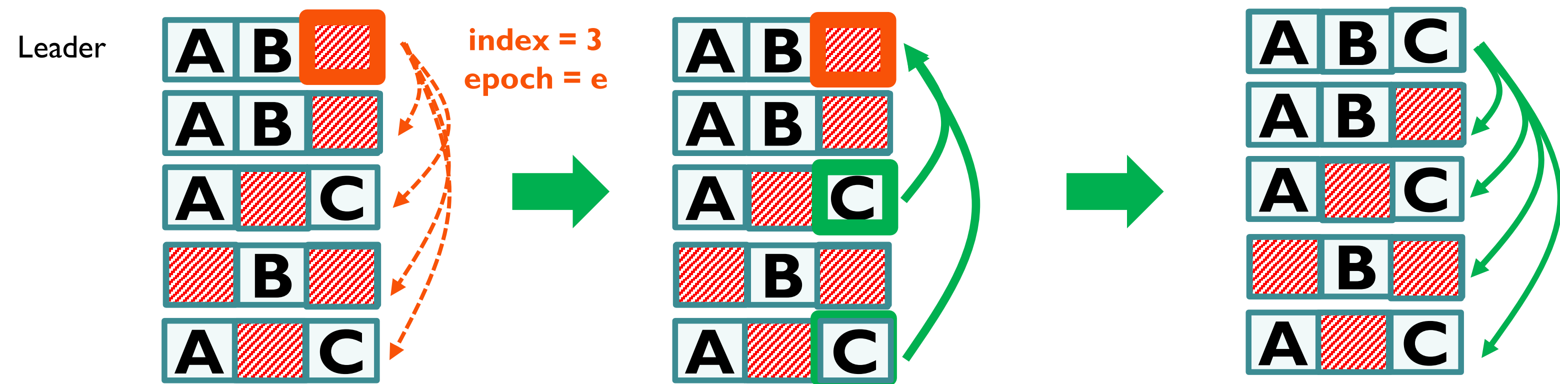


2018 Storage Developer Conference. © University of Wisconsin - Madison. All Rights Reserved.

Leader Log Recovery

Fixing the leader is the tricky part

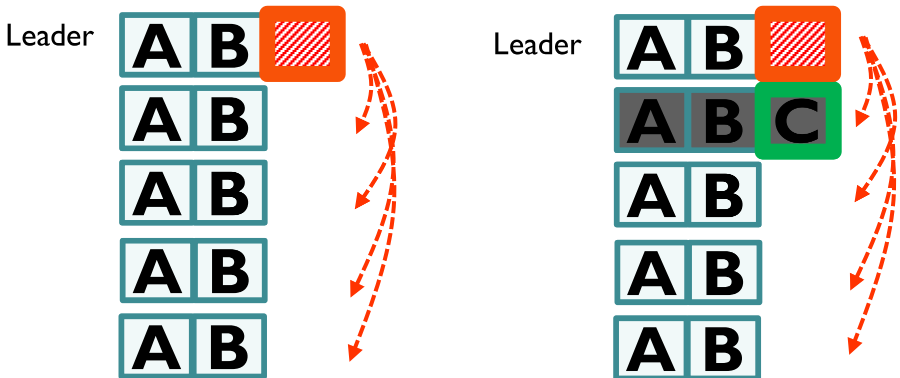
First, a simple case: some follower has the entry intact



2018 Storage Developer Conference. © University of Wisconsin - Madison. All Rights Reserved.

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

2018 Storage Developer Conference. © University of Wisconsin - Madison. All Rights Reserved.

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

2018 Storage Developer Conference. © University of Wisconsin - Madison. All Rights Reserved.

Evaluation

We apply CTRL in two systems

LogCabin

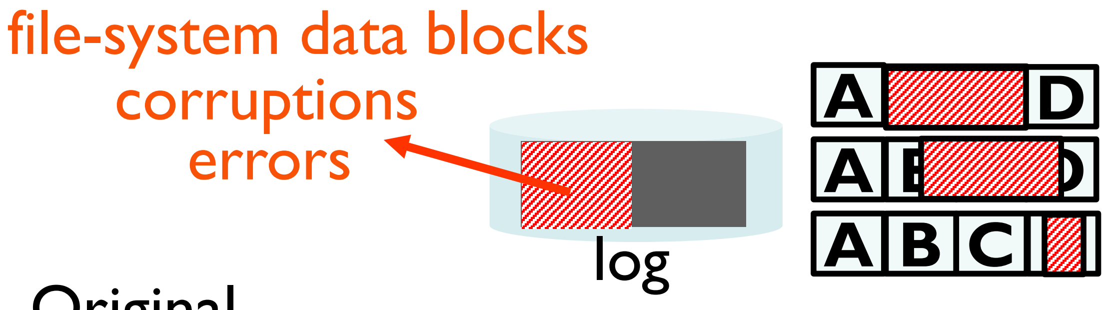
- based on Raft

ZooKeeper

- based on ZAB

2018 Storage Developer Conference. © University of Wisconsin - Madison. All Rights Reserved.

Reliability Experiments Example



Original

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

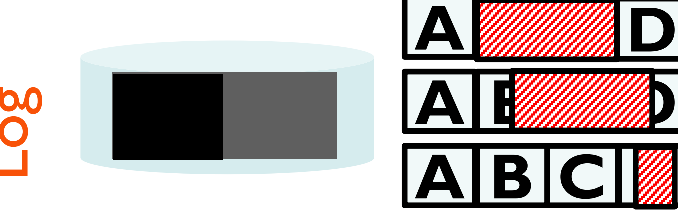
CTRL

- corruptions and errors: always safe and available

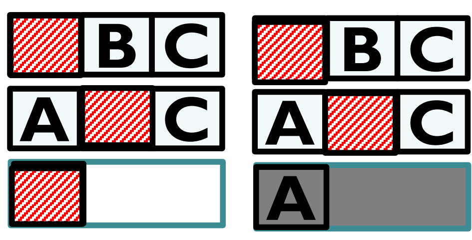
2018 Storage Developer Conference. © University of Wisconsin - Madison. All Rights Reserved.

Reliability Experiments Summary

Log



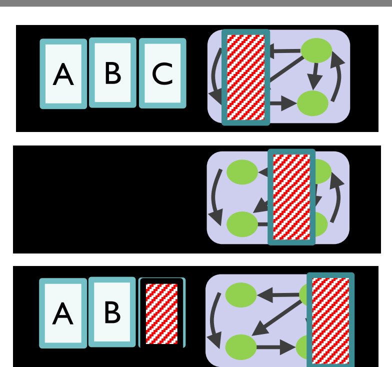
all possible combinations (for thoroughness)

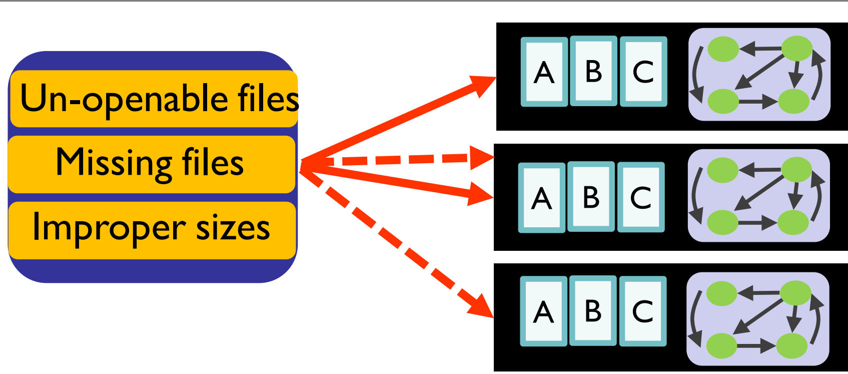


FS data blocks

Targeted entries

Lagging and crashed





2018 Storage Developer Conference. © University of Wisconsin - Madison. All Rights Reserved.

Reliability Results Summary

Original systems

- unsafe or unavailable in many cases

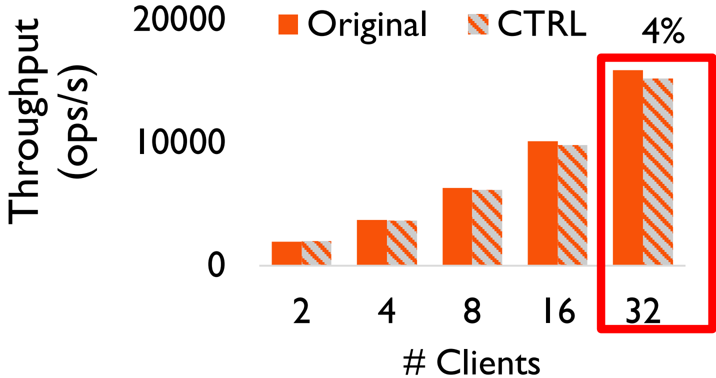
CTRL versions

- safe always and highly available
- correctly unavailable in some cases (when all copies are faulty)

2018 Storage Developer Conference. © University of Wisconsin - Madison. All Rights Reserved.

Update Performance (SSD)

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



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

2018 Storage Developer Conference. © University of Wisconsin - Madison. All Rights Reserved.

Part-2 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

2018 Storage Developer Conference. © University of Wisconsin - Madison. All Rights Reserved.

Summary

Part-1: measure how distributed storage systems react to storage faults such as corruption and errors

Main result: redundancy does not imply fault tolerance, some fundamental root causes

Part-2: build a new recovery protocol for RSM, CTRL, safe and available, little overheads

2018 Storage Developer Conference. © University of Wisconsin - Madison. All Rights Reserved.

Conclusions

Obvious things we take for granted in distributed systems: redundant copies will help recover bad data or redundancy → reliability are surprisingly hard to achieve

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

 © University of Wisconsin - Madison. All Rights Reserved.

Research to Practice

Cords: Storage Corruption and Errors Tool

- *errfs* – a fuse FS, a similar FS now part of Jepsen
- similar methods applied by a few companies now (e.g., CockroachDB)

Available @ http://research.cs.wisc.edu/adsl/Software/

Related papers @ http://research.cs.wisc.edu/adsl/Publications/

Joint work with *Aishwarya Ganesan, Andrea Arpaci-Dusseau, and Remzi Arpaci-Dusseau*

Thank you!

2018 Storage Developer Conference. © University of Wisconsin - Madison. All Rights Reserved.

Backup Slides

2018 Storage Developer Conference. © University of Wisconsin - Madison. All Rights Reserved.

Crashing - Common Local Reaction

Many systems that reliably detect fault simply crash on encountering faults

Block Corruption during Read Workloads

MongoDB	ZooKeeper
collections.header	epoch
collections.metadata	epoch_tmp
collections.data	myid
index	log.transaction_head

L F   L F

☒ Crash

Crashing leads to reduced redundancy and imminent unavailability

Persistent fault -- Requires manual intervention

Redundancy underutilized!

2018 Storage Developer Conference. © University of Wisconsin - Madison. All Rights Reserved.

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

2018 Storage Developer Conference. © University of Wisconsin - Madison. All Rights Reserved.

Protocol-Oblivious: Crash

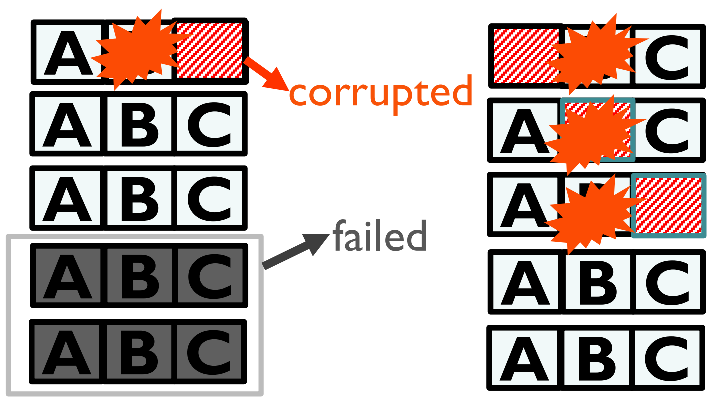
Crash

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)

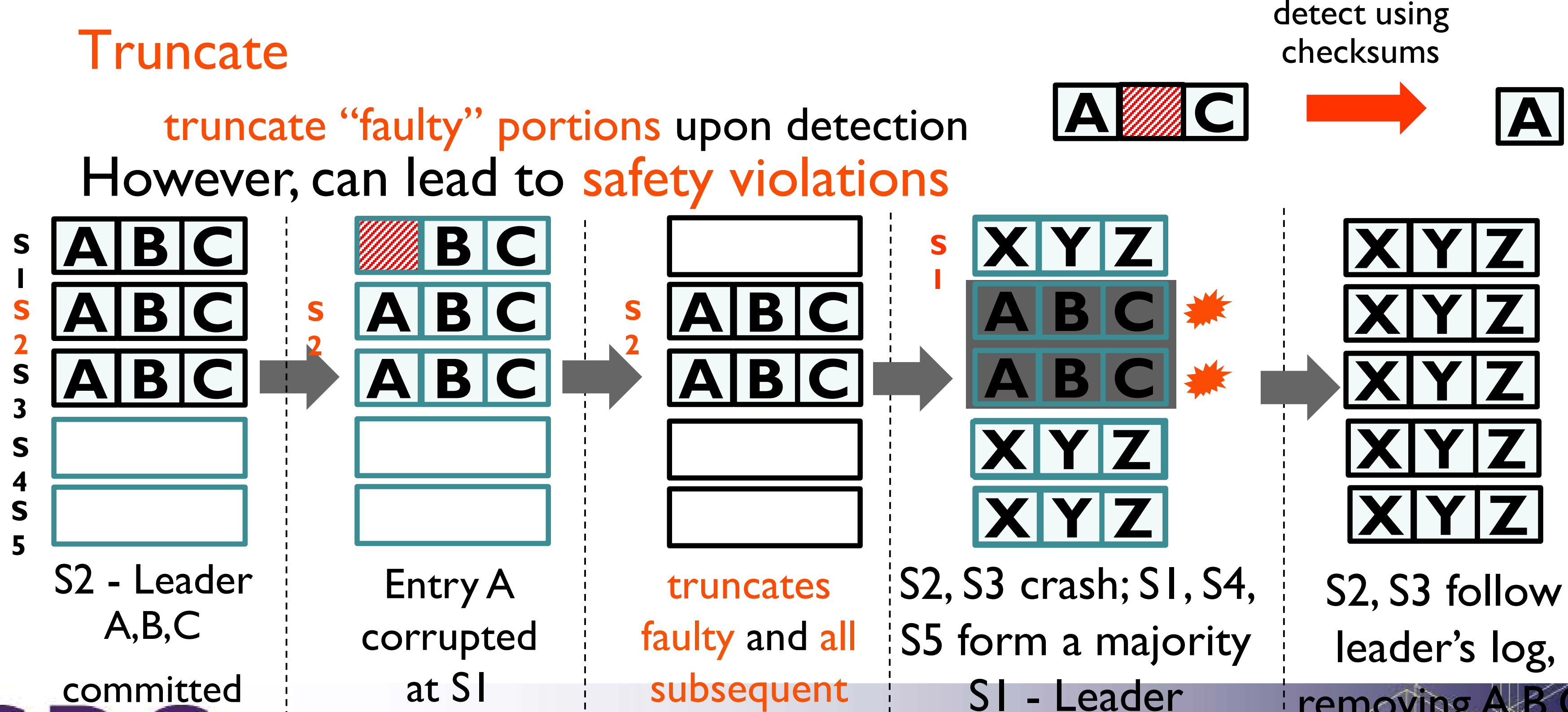
2018 Storage Developer Conference. © University of Wisconsin - Madison. All Rights Reserved.

Protocol-Oblivious: Truncate

Truncate

truncate "faulty" portions upon detection

However, can lead to safety violations



2018 Storage Developer Conference. © University of Wisconsin - Madison. All Rights Reserved.

Recovery Approaches Summary

Class	Approach	Safety	Availa-bility	Perform-ance	No intervention	No extra nodes	Fast recovery	Low complexity
Protocol-oblivious	NoDetection	✗	✓	✓	✓	✓	NA	✓
	Crash	✓	✗	✓	✗	✓	NA	✓
	Truncate	✗	✓	✓	✓	✓	✗	✓
	DeleteRebuild	✗	✓	✓	✗	✓	✗	✓
Protocol-aware	MarkNonVote [1]	✗	✗	✓	✓	✓	✗	✓
	Reconfigure [2]	✓	✗	✓	✗	✗	✗	✓
	Byzantine FT	✓	✗	✗	✓	✗	NA	✗
	CTRL	✓	✓	✓	✓	✓	✓	✓

[1] Chandra et al., PODC '07 [2] Bolosky et al., NSDI '11

2018 Storage Developer Conference. © University of Wisconsin - Madison. All Rights Reserved.