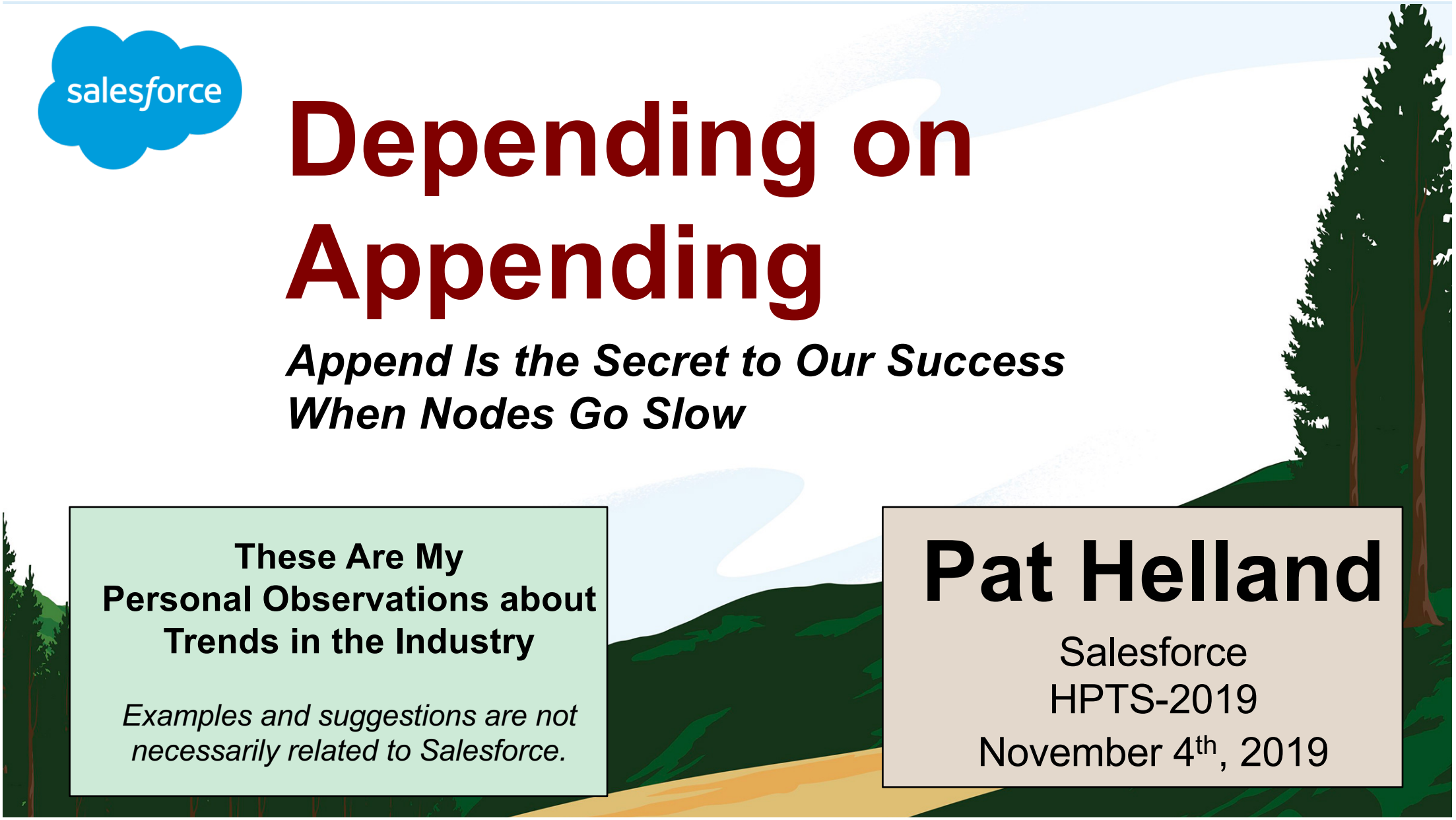


Depending on Appending

Append Is the Secret to Our Success When Nodes Go Slow

These Are My Personal Observations about Trends in the Industry

Examples and suggestions are not necessarily related to Salesforce.

Pat Helland

Salesforce

HPTS-2019

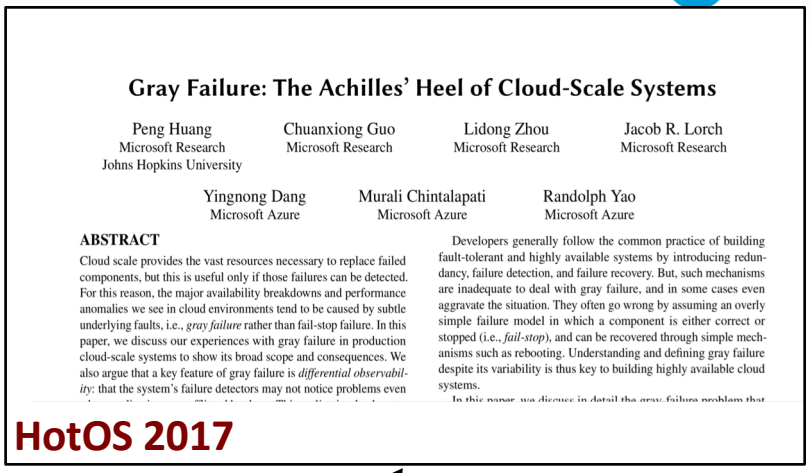
November 4th, 2019

Outline

- Introduction
- Examples of Append in Distributed Systems
- Appending Fast While Failing Gray
- Databases and Gray Failures
- Conclusion

50 Shades of Gray Failures…

- **Gray failures:**
  - Servers & network routers slow down
  - Software gets unresponsive (slow)
    - Retries, garbage collection and more…
  - Single node gray failures → wedge the cluster!



Gray Failure: The Achilles' Heel of Cloud-Scale Systems

Peng Huang, Microsoft Research, Johns Hopkins University
Chuanxiong Guo, Microsoft Research
Lidong Zhou, Microsoft Research
Jacob R. Lorch, Microsoft Research
Yingnong Dang, Microsoft Azure
Murali Chintalapati, Microsoft Azure
Randolph Yao, Microsoft Azure

ABSTRACT

Cloud scale provides the vast resources necessary to replace failed components, but this is useful only if those failures can be detected. For this reason, the major availability breakdowns and performance anomalies we see in cloud environments tend to be caused by subtle underlying faults, i.e., *gray failure* rather than fail-stop failure. In this paper, we discuss our experiences with gray failure in production cloud-scale systems to show its broad scope and consequences. We also argue that a key feature of gray failure is *differential observability*: that the system's failure detectors may not notice problems even

Developers generally follow the common practice of building fault-tolerant and highly available systems by introducing redundancy, failure detection, and failure recovery. But, such mechanisms are inadequate to deal with gray failure, and in some cases even aggravate the situation. They often go wrong by assuming an overly simple failure model in which a component is either correct or stopped (i.e., *fail-stop*), and can be recovered through simple mechanisms such as rebooting. Understanding and defining gray failure despite its variability is thus key to building highly available cloud systems.

HotOS 2017

Gray failures seem to be happening more often

"As cloud systems increase in scale and complexity,
***gray failure** becomes more common"*

- **Gray failures are complex to deal with**
  - Did the server not answer because it's dead?
  - Did the server not answer because it's slow?

Servers and Routers <u>WILL Go Slow</u>
When We Least Expect It

Can We Keep the System Fast?

Latency and Transactional Consistency

- **Latency:**
  - Batch systems don't worry about latency
    - Intermittent slow latency OK for batch
  - Online systems care a lot about latency!
    - Measured SLA
    - Human beings hate variable latency!

- **Transactional consistency and bounded latency**
  - Transactional consistency depends on order and what's gone before
  - Most systems funnel their consistency through a single server or servers
  - ***Serializability: Making changes look as-if they happen one at a time***

Can we have **serial order without delays?**

Can we do this while servers run slow??

Tricks Up Our Sleeve…

Pre-allocating Where to Log

To Start Recovery, a DB Needs to Find Its Log Files

The Identity of Log Files Is Usually Managed Centrally

Central Management May Stall from Gray Failure

Pre-allocate Log Locations

Quorum Appends

Log Replicas May Stall

Accept a Log Write when a Quorum of the Replicas Have Seen the Append

Use Quorum Logic to Determine the End of the Log

Pool of Compute

A Pool of SQL Engines

Route Requests to Any Available Engine

Like Microservices

SQL Engines Fight to Append

Order Depends on Appends

Optimistic Concurrency: Winners & Losers

Outline

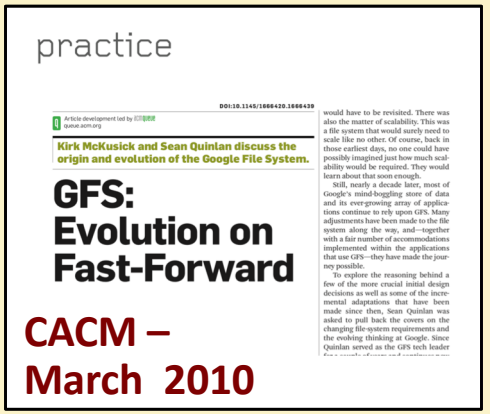
- Introduction
- Examples of Append in Distributed Systems
- Appending Fast While Failing Gray
- Databases and Gray Failures
- Conclusion

Multi-Writer GFS, HDFS, and Cosmos (Scope)

Multi-Writer GFS (Google File System)

- Client → primary → secondary → tertiary
- Multiple concurrent writes AND primary failure!
  - Different requests land at secondary and primary
  - *Unrepeatable reads after a primary crash!*

No coordination of write order when failure happens!


practice

Kirk McKusick and Sean Quinlan discuss the origin and evolution of the Google File System.

GFS: Evolution on Fast-Forward

CACM – March 2010

HDFS (Hadoop Distributed File System)

- Distributed file system designed for Hadoop (MapReduce)
- Constrained to single writer for each file

Write order is defined by client

Cosmos (Scope) – Distributed File System used beneath Bing

- Primary defines order unless it dies
- Secondary consults master CSM and truncates block size

Write order is defined by primary unless failure then defined by master

Latency Bounding: a Log Tale

- **Apache Bookkeeper: A log for DBs or similar servers**
  - Database servers write to log files (*ledgers*) → Each write is called an *entry*
  - Each ledger is kept on three servers called *bookies*
- **Each write from the DB is sent to 3 bookies and waits for 2 acks**
  - Each write has a ledger-number and an entry-number
  - *When the entry (and all before it) are each ack-ed by 2 bookies, it's durable*

Ledger-IDs AND Entry-IDs are supplied by the database to the store

Ledger-IDs *must be remembered in a log-window*

Entry-IDs *are assigned by the DB by counting*

Recovery explores a set of Ledgers (a log-window) and finds the last entry by reading to the end

Tolerates Slow Bookies

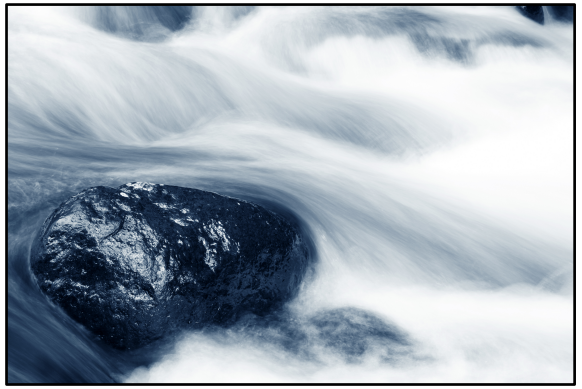
Flows Around
a Slow Bookie
Like a River
Around
a Big Rock

Who Defines the Location of the Append?

	Who Defines Where an Append Lands?	What Happens If "Append-Picker" Fails?	What Happens If the Data Node Slows Down?	What Happens If "Append-Picker" Slows Down?
GFS	Primary Data Node	Corruption	Slow Appends	Slow Appends *(Primary Data Node)*
HDFS	Client	Client Crashes → File Append Stops	Slow Appends	Slow Appends *(Client)*
Cosmos	Primary Data Node	Master Decides Next Append Location	Slow Appends	Slow Appends *(Primary Data Node)*
Bookkeeper	Client *(the DB)*	DB Crashes → File Append Stops	***No Slow Down Log Append Continues***	Slow Appends *(DB Client Is Slow)*

Outline

- Introduction
- Examples of Append in Distributed Systems
- Appending Fast While Failing Gray
- Databases and Gray Failures
- Conclusion

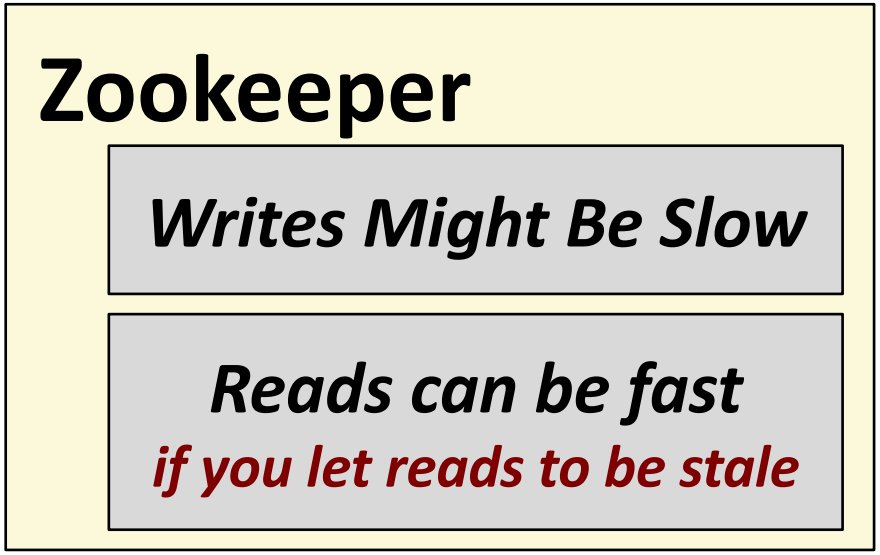


Working Around Obstacles…

Planning for Delays when Using Zookeeper

- **Zookeeper works with a leader and a leader election**
  - The leader does the work and the followers track a slightly delayed version
  - This is fast when the follower is healthy
  - When a leader fails: Leadership election can take a while
  - *When a leader goes slow (Gray Failure) all progress writing gets very slow!*

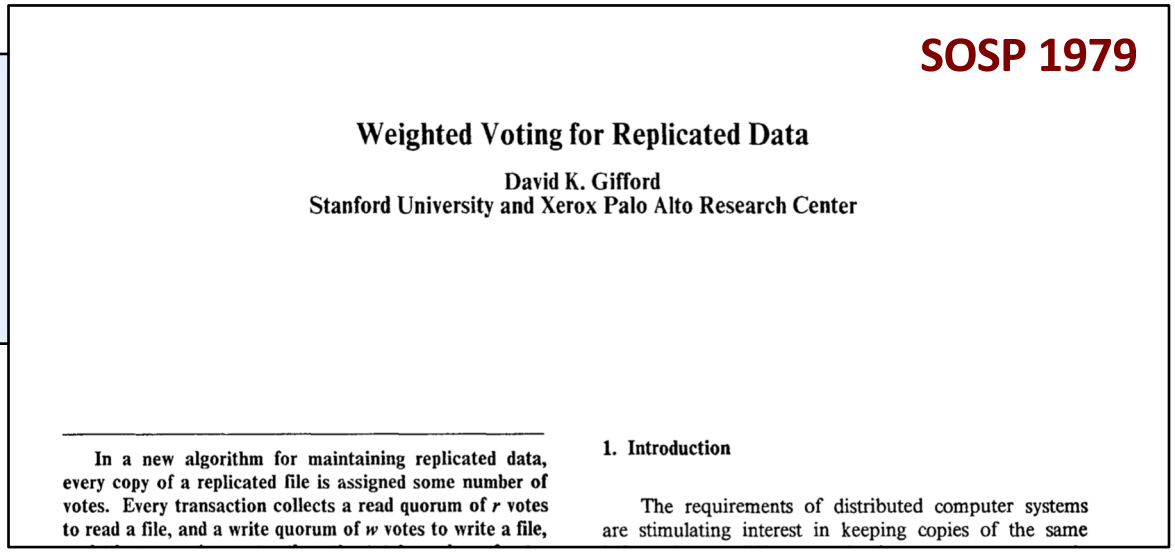
- **Assumptions about using Zookeeper**
  - Writes are sometimes slow
    - Gray Failures may stall writes
  - Stale reads can be fast if you:
    - Read from one of the followers
    - If response is slow, retry to another follower



Love the Ones You're With (or Most of Them)

Gifford's Algorithm:

R + W > N



SOSP 1979

Weighted Voting for Replicated Data

David K. Gifford
Stanford University and Xerox Palo Alto Research Center

In a new algorithm for maintaining replicated data, every copy of a replicated file is assigned some number of votes. Every transaction collects a read quorum of r votes to read a file, and a write quorum of w votes to write a file,

1. Introduction

The requirements of distributed computer systems are stimulating interest in keeping copies of the same

- **Smart logs can assign append location**
  - Log append:
    - Writing client requests append
    - Next log append will be after, not necessarily immediately after last append
  - Quorum log append:
    - Only when W of the log servers agree on the target location for an append will it be added to the log

Append: Logging Client Writes to "N" and Waits for "W"

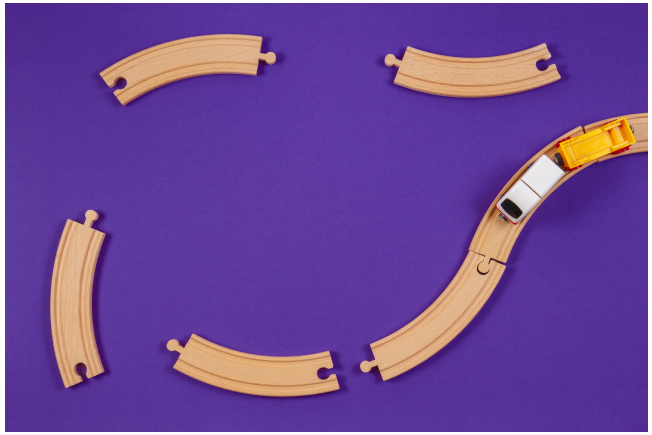
Append: Logging Servers Coordinate Until "W" Servers Agree on the Log-Address for the Append

Knowing WHERE to Log Can Be a Problem

- **Allocation of log-files is a challenge**
  - Typically, this is centralized in the system
  - As the system becomes distributed, this is typically centralized in Zookeeper or some equivalent system
- **Centralized selection of log files can stall !**
  - Just picking where to write can have gray failures!
- **I need to log and know where I'm logging!**
  - The files of the log must be allocated
  - The location of the log files must be recorded!
  - Recording in Zookeeper might stall
    - Sometimes for minutes!
  - **Solution: Preallocate hours of log files**

Where Do I Go Next?

There Must Be a Plan!



Jittery Preallocation and Stale Reads

- **Preallocate a window of log file names**
  - This can be done using Zookeeper or some equivalent
  - Sometimes, this will go slow as the ZK leader has a Gray Failure
  - Preallocate a lot of files to support hours of logging
- **On crash restart do a stale read of the window of log file names**
  - It's OK if it's not the latest
  - Do a binary search of the log files in the log window
  - Find the latest log file name with log contents in it

A Stale Log Window (List of Log Files) Can Work Well for Crash Recovery

After a Crash, Explore the Log Window for the Last Written File

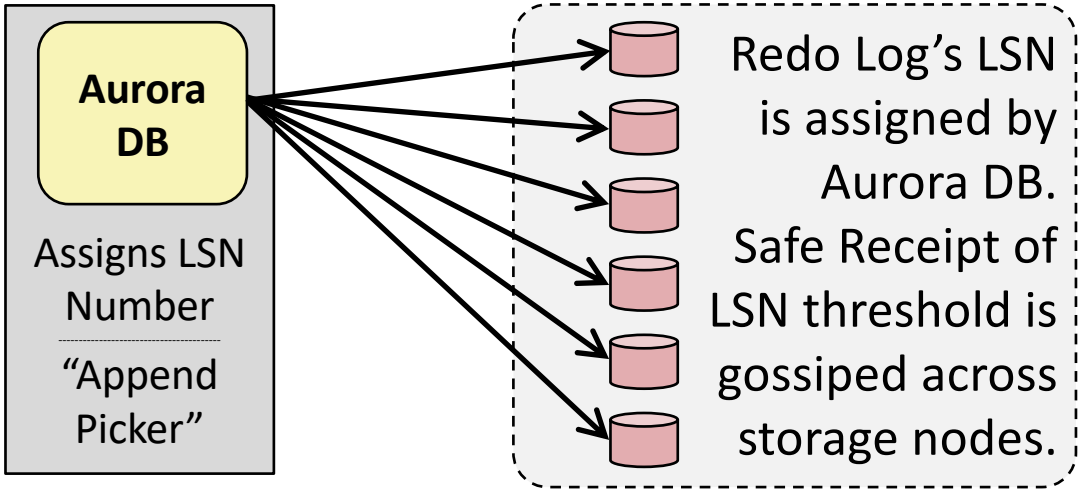
A Strongly Consistent List *(e.g. Kept in Zookeeper)* ***Can Be Used***

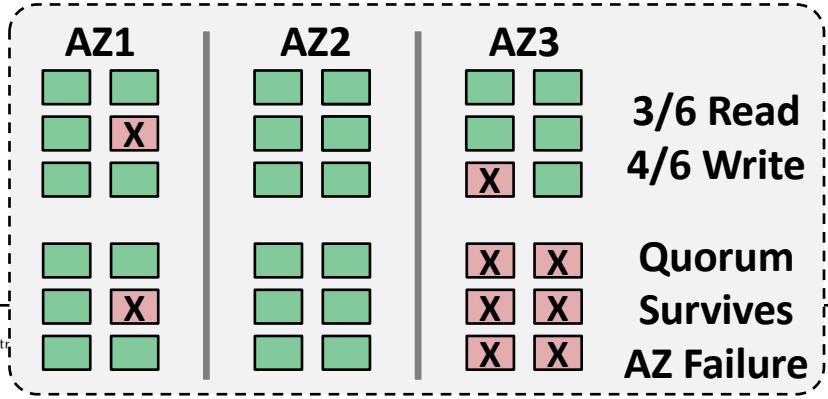
Outline

- Introduction
- Examples of Append in Distributed Systems
- Appending Fast While Failing Gray
- Databases and Gray Failures
- Conclusion

Amazon Aurora: Tolerates Slow Storage (e.g. Logs)

- **AWS Aurora is a database for the cloud (supporting MySQL and Postgres)**
  - 3 Availability Zones (AZs) with 2 storage nodes on each AZ
- **DB engine ships redo logs to 6 storage nodes and waits for 4 to answer**
  - 4 out of 6 tolerates AZ+1 failures
- **Redo log's LSN used to coordinate quorum delivery across storage nodes**





SIGMOD 2018

Amazon Aurora: On Avoiding Distributed Consensus for I/Os, Commits, and Membership Changes

Alexandre Verbitski, Anurag Gupta, Debanjan Saha, James Corey, Kamal Gupta
Murali Brahmadesam, Raman Mittal, Sailesh Krishnamurthy, Sandor Maurice
Tengiz Kharatishvilli, Xiaofeng Bao
Amazon Web Services

ABSTRACT

Amazon Aurora is a high-throughput cloud-native relational database offered as part of Amazon Web Services (AWS). One of the more novel differences between Aurora and other relational databases is how it pushes redo processing to a multi-tenant scale-out storage service, purpose-built for Aurora. Doing so reduces networking

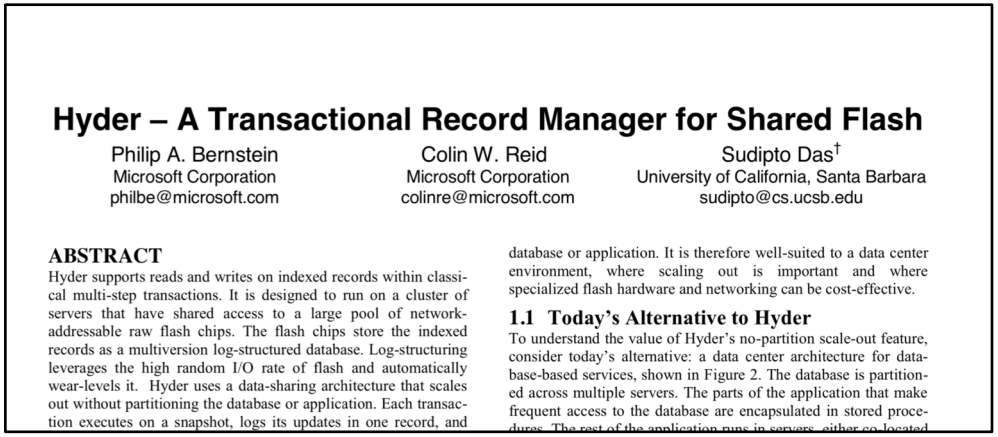
database instances in RDS led to the design requirements for Aurora, a high-throughput cloud-native relational database.

In our earlier paper [12], we provided an overview of the design considerations behind Aurora. A key contribution of that paper is to show that, on a fleet-wide basis, it is insufficient to treat failures

Tolerates Slow Storage Servers
Does NOT Tolerate a Slow Aurora DB

Hyder: Tolerates Slow Databases (not Logs)

- **Hyder is a database system with many DB servers sharing a single DB log**
  - **Snapshot:** Start TX based on point-in-time
  - **Intention:** Log the hoped for TX'I changes
  - **Meld:** Check for conflicts; commit or abort
    - Only need to check in the **conflict-zone** since the snapshot
  - **Commit or Abort** by logging the state



Hyder – A Transactional Record Manager for Shared Flash

Philip A. Bernstein
Microsoft Corporation
philbe@microsoft.com

Colin W. Reid
Microsoft Corporation
colinre@microsoft.com

Sudipto Das†
University of California, Santa Barbara
sudipto@cs.ucsb.edu

ABSTRACT

Hyder supports reads and writes on indexed records within classical multi-step transactions. It is designed to run on a cluster of servers that have shared access to a large pool of network-addressable raw flash chips. The flash chips store the indexed records as a multiversion log-structured database. Log-structuring leverages the high random I/O rate of flash and automatically wear-levels it. Hyder uses a data-sharing architecture that scales out without partitioning the database or application. Each transaction executes on a snapshot, logs its updates in one record, and

database or application. It is therefore well-suited to a data center environment, where scaling out is important and where specialized flash hardware and networking can be cost-effective.

1.1 Today's Alternative to Hyder

To understand the value of Hyder's no-partition scale-out feature, consider today's alternative: a data center architecture for database-based services, shown in Figure 2. The database is partitioned across multiple servers. The parts of the application that make frequent access to the database are encapsulated in stored procedures. The rest of the application runs in servers, either co-located

- **Hyder tolerates slow DB servers reading/writing the same shared log**
  - If one DB server is slow, the rest will continue → App requests retry to other servers

- **But what if log servers are slow???**
  - A slow log can mean a slow Hyder DB!

Can We Do Hyder with Quorum Logs?

- **Latency bounding for the log requires log writes be written in parallel**
  - No single log replica can define the order (or it can stall the write)
  - No single DB replica can define the order (or it can stall the write)
- **Hyder requires the intentions written to the log be in order**
  - Writes must be ordered by quorum
- **Each of the N log replicas must:**
  - **Receive intention** requests from many databases
  - **Propose an order** for the intentions and negotiate with the other replicas
    - Reordering as necessary to get an agreed order for the intentions
  - **Confirm quorum** with "W" total replicas of the log
  - **Acknowledge transaction commits** when *meld succeeds*

A Bit Like Herding Cats!

Outline

- Introduction
- Examples of Append in Distributed Systems
- Appending Fast While Failing Gray
- Databases and Gray Failures
- Conclusion

Depending on Appending

Gray failures will be part of our lives

System hardware and software will sometimes “go slow”!

Strongly consistent SQL databases are important

Important to work with low latency and without stalls for OLTP !

Append is the Secret Sauce to Avoid Stalls

Ensure multiple writers are appending and any can stall
Ensure multiple log replicas receive appends and any can stall

In the future, we will be

Depending on Appending

For an increasing number of things….

That's All Folks!!