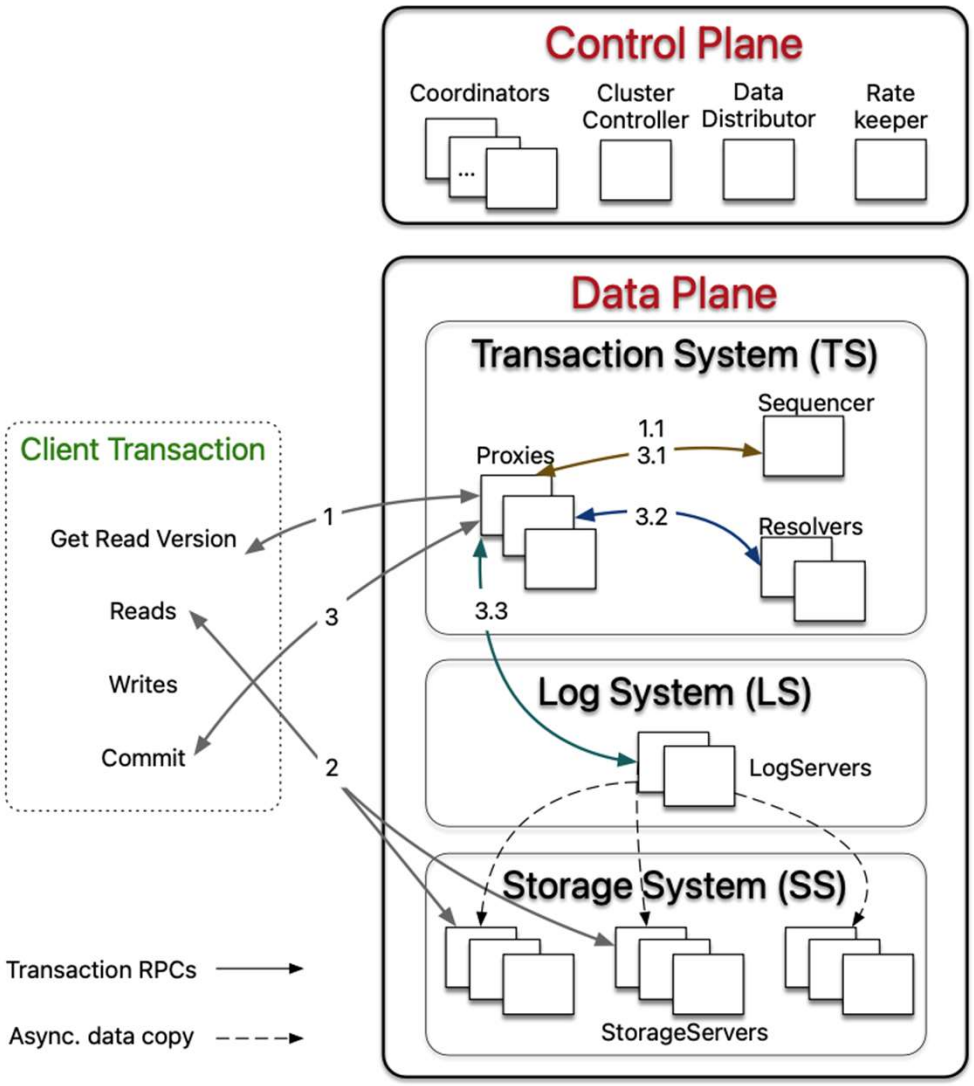
Cloud and Cluster Data Management

FOUNDATIONDB: A DISTRIBUTED UNBUNDLED TRANSACTIONAL KEY VALUE STORE

FDB Architecture



- **Control Plane:**
  - responsible for persisting critical system metadata: configuration of trxn systems, on Coordinators.
- **Data Plane:**
  - FDB uses an unbundled arch.
    - Distributed TS: performs in-memory trxn processing
    - Log system (LS): stores Write-Ahead-Log (WAL) for TS
    - Storage system (SS): for storing data and servicing reads

Figure Credit: "FoundationDB: A Distributed Unbundled Transactional Key Value Store, by Zhou J, et al. 2021"

Support Strict Serializability

- FDB implements Serializable Snapshot Isolation (SSI) by combining OCC with MVCC
- The sequencer's process for providing read/commit versions to client trxns defines a serial history for trxns & serve as Log Sequence Number (LSN)
- Because Tx observes the results of all prev. committed trxns, FDB achieves Strict Serializability.
- Sequencer returns both LSN and prev. LSN to Resolvers and LSs
  - So they can serially process trxns in order of LSNs
- SSs pull log data from LSs in increasing LSNs.

Lock-Free Conflict Detection Algorithm

Each Resolver maintains a history lastCommit of recently modified key ranges by committed trxns & their corresponding commit versions

Each Tx commit request comprises two sets:
Rw: modified key ranges
Rr: set of read key ranges

hm 1: Check conflicts for transaction T_x.

Require: *lastCommit*: a map of key range → last commit version

```
1 for each range ∈ R_r do
2     ranges = lastCommit.intersect(range)
3     for each r ∈ ranges do
4         if lastCommit[r] > T_x.readVersion then
5             return abort;
  // commit path
6 for each range ∈ R_w do
7     lastCommit[range] = T_x.commitVersion;
8 return commit;
```

Rr is checked against modified read ranges of conc. committed trxns (prevents phantom read)

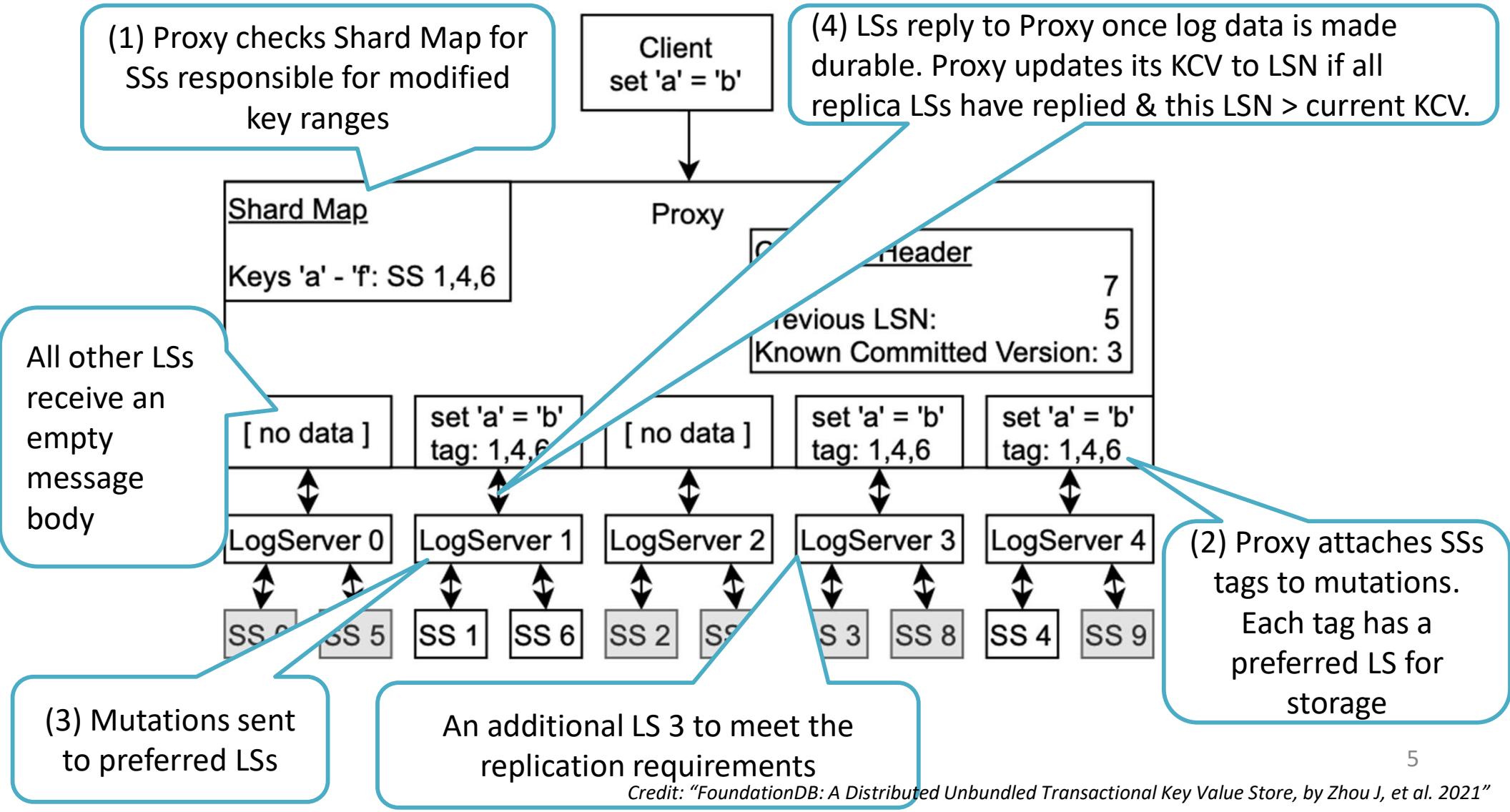
If no R-W conflict, Resolver admits trxn for commit & updates the list of modified key ranges with the write set

- Entire key space is divided among Resolvers so this algorithm may be performed in parallel.
- Trxn can commit only when all Resolvers admit the trxn. Otherwise, it is aborted.

Credit: "FoundationDB: A Distributed Unbundled Transactional Key Value Store, by Zhou J, et al. 2021"

Logging Protocol

Log messages are broadcasted to all LSs once a Proxy decides to commit a trxn



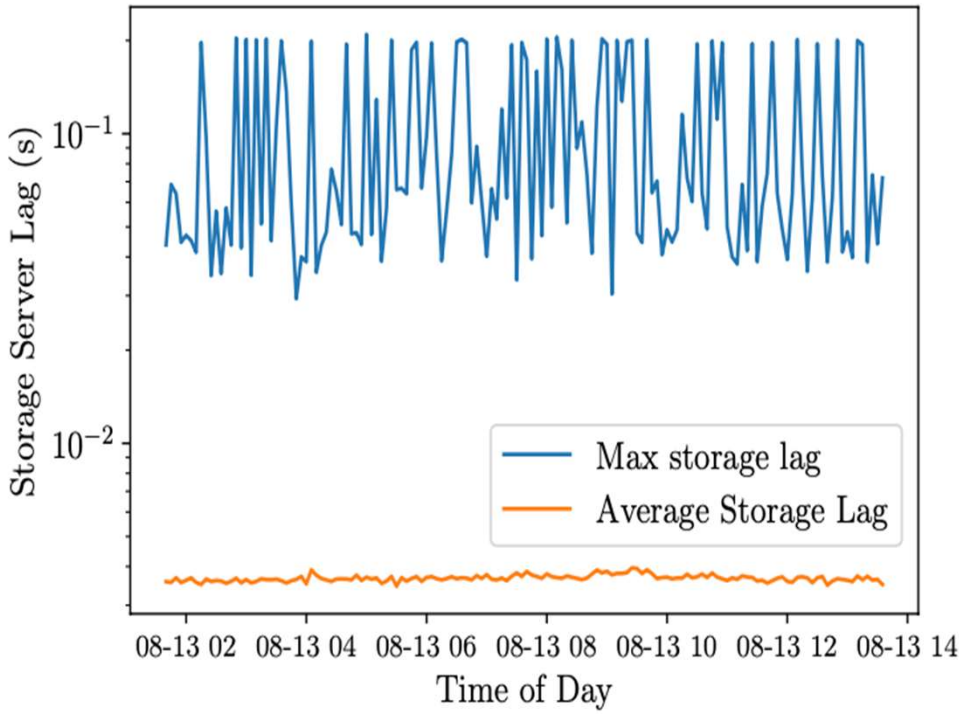
Credit: "FoundationDB: A Distributed Unbundled Transactional Key Value Store, by Zhou J, et al. 2021"

Participation Question

Why do some LogServers receive an empty message body from the Proxy?

Lag from StorageServer to LogServer

- Shipping redo log from LS to SS is performed in the background
- SSs aggressively fetch redo logs from LSs before they are durable on the LS
  - very low latency for serving multi-version reads
- Lag from SS to LS for 12-hour period:
  - 99.9 % of the average: 3.96ms
  - Max delay: 208.6ms
- When client read requests reach SS, requested version (i.e., latest committed data) is usually already available

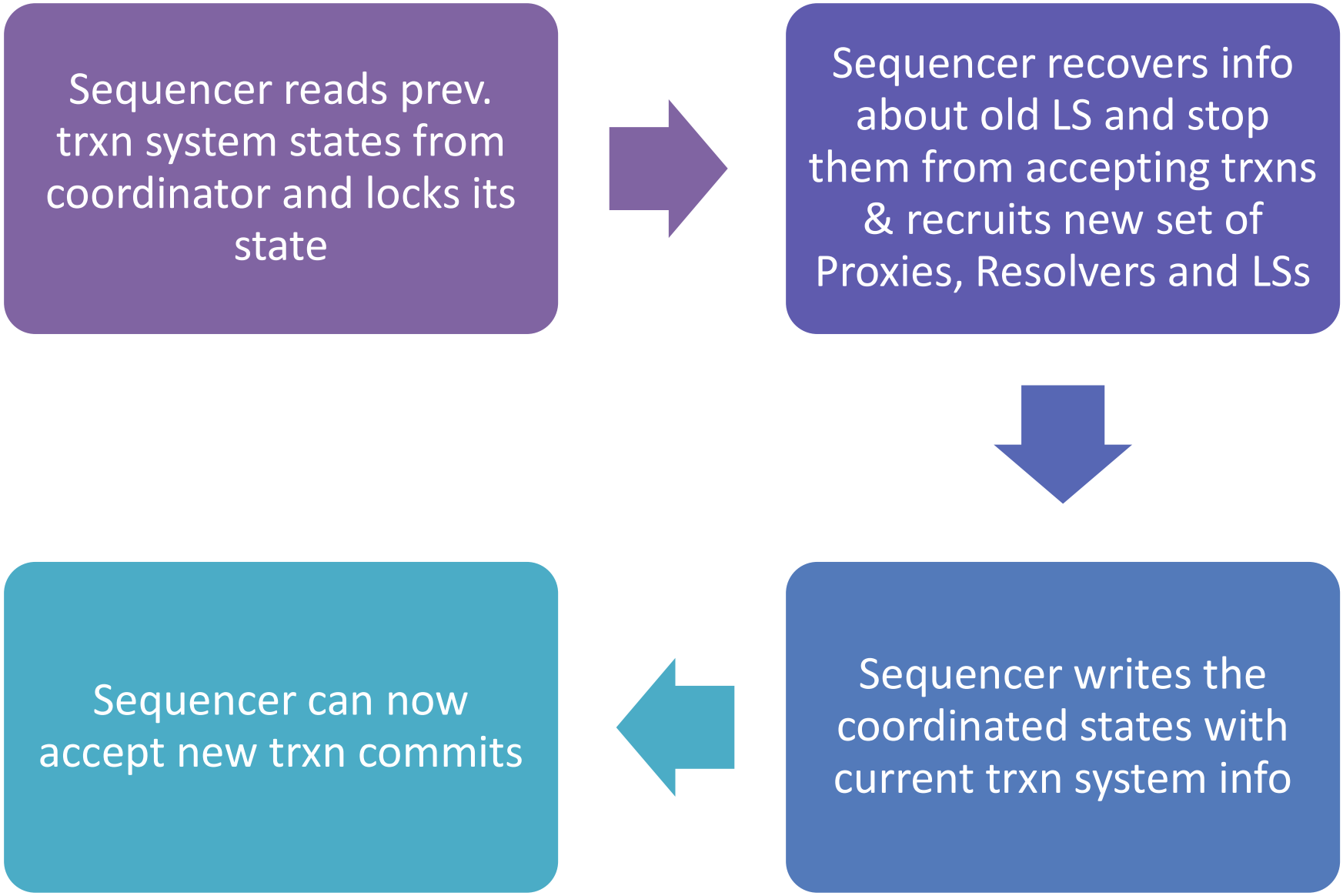


Credit: "FoundationDB: A Distributed Unbundled Transactional Key Value Store, by Zhou J, et al. 2021"

Transaction System Recovery

- FDB recovery is cheap: no checkpoint, no need to re-apply redo or undo log
- FDB decouples redo log processing from recovery
  - SS always pull logs from LS and apply them in the background
- Recovery starts when a failure is detected.
  - Recruits new trxn system which can accept trxns before all data on old LSs is processed
  - **The recovery only needs to determine the end of redo log and re-applying the log is performed asynchronously by SS**

Recovery Process



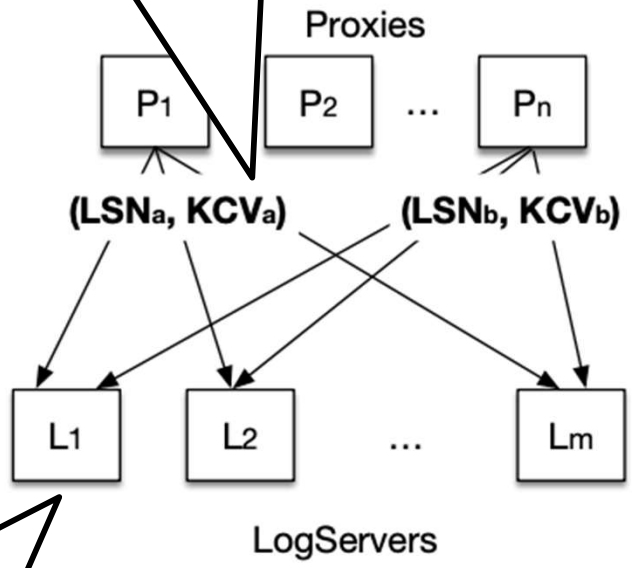
Recovering the LogServers

- Proxies and Resolvers are stateless, no extra work needed for recovery
- LSs save log of committed trxns, so all prev. committed trxns must be durable and retrievable by SS
  - i.e., the log of any trxn the Proxies may have sent back a commit response must be persisted in multiple LS (satisfying replication degree)
- The essence of old LSs recovery is determining the end of redo log (Recovery Version)
- Rolling back undo log is discarding any data after RV in the old LSs & SSs

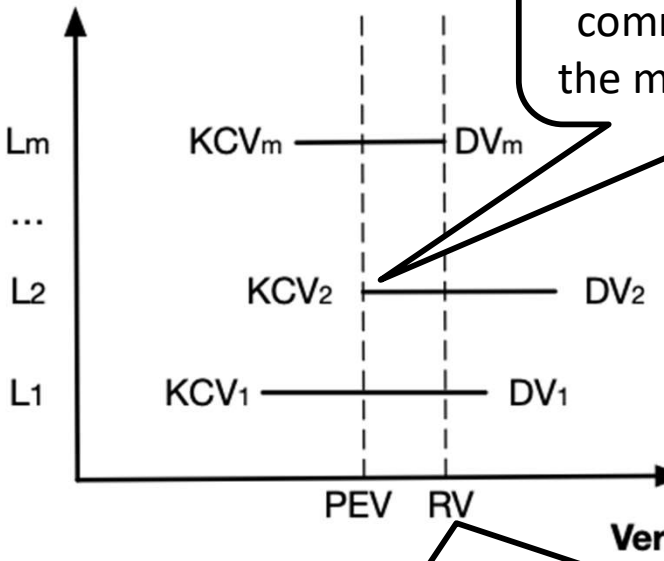
Determining RV and PEV

Sequencer attempts to stop all m old LSs, each LS response contains their KCV and DV.

Each Proxy request to LS sends its KCV (max LSN committed by this proxy)



Sequencer knows the prev. epoch has committed trxns up to the max of all KCVs(PEV)



RV: Recovery Version
PEV: Prev. Epoch's end Version

Each LS keeps the max KCV and Durable Version (DV: max persisted LSN)

For current epoch, its start version is $PEV + 1$ and the Sequencer chooses the min of all DVs to be the RV.

Logs in $[PEV + 1, RV]$ are copied from prev. epoch's LSs to current ones for healing the replication degree in case of LS failures

Credit: "FoundationDB: A Distributed Unbundled Transactional Key Value Store, by Zhou J, et al. 2021"

Replications in FDB

FDB uses various replication strategies for different data to tolerate f failures.

- **Metadata replication**. System metadata of the control plane is stored on Coordinators using Active Disk Paxos.
  - Metadata can be recovered if majority of Coordinators are alive
- **Log replication**. When a Proxy writes logs to LSs, each sharded log record is synchronously replicated on $k = f + 1$ LSs.
  - Proxy sends commit response to client only when all k replied with success
- **Storage replication**. Every shard is asynchronously replicated to $k = f + 1$ SSs (team)
  - A SS usually hosts several shards so that its data is evenly distributed across many teams.
  - A failure of a SS triggers DataDistributor to move data from teams containing the failed process to other healthy teams.

Other Optimizations

- **Transaction batching.** Proxy writes committed trxns in batch to LS
  - Reduces the number of calls to obtain CV from sequencer, Proxy commits 10Ks trxs/s without significantly impacting Sequencer's performance
  - batching degree is adjusted dynamically: Shrinking when the system is lightly load to improve latency, increasing when the system is busy to sustain high commit throughput
- **Atomic operations.** atomic add, bitwise "and", compare-and-clear, and set-versionstamp (sets part of the key or part of the value to be the transaction's commit version)
  - enable a trxn to write a data item without reading its value, saving a round-trip time to SSs.
  - eliminate RW conflicts with other atomic operations on the same data item (WR conflicts can still happen)

Geo-replication and failover

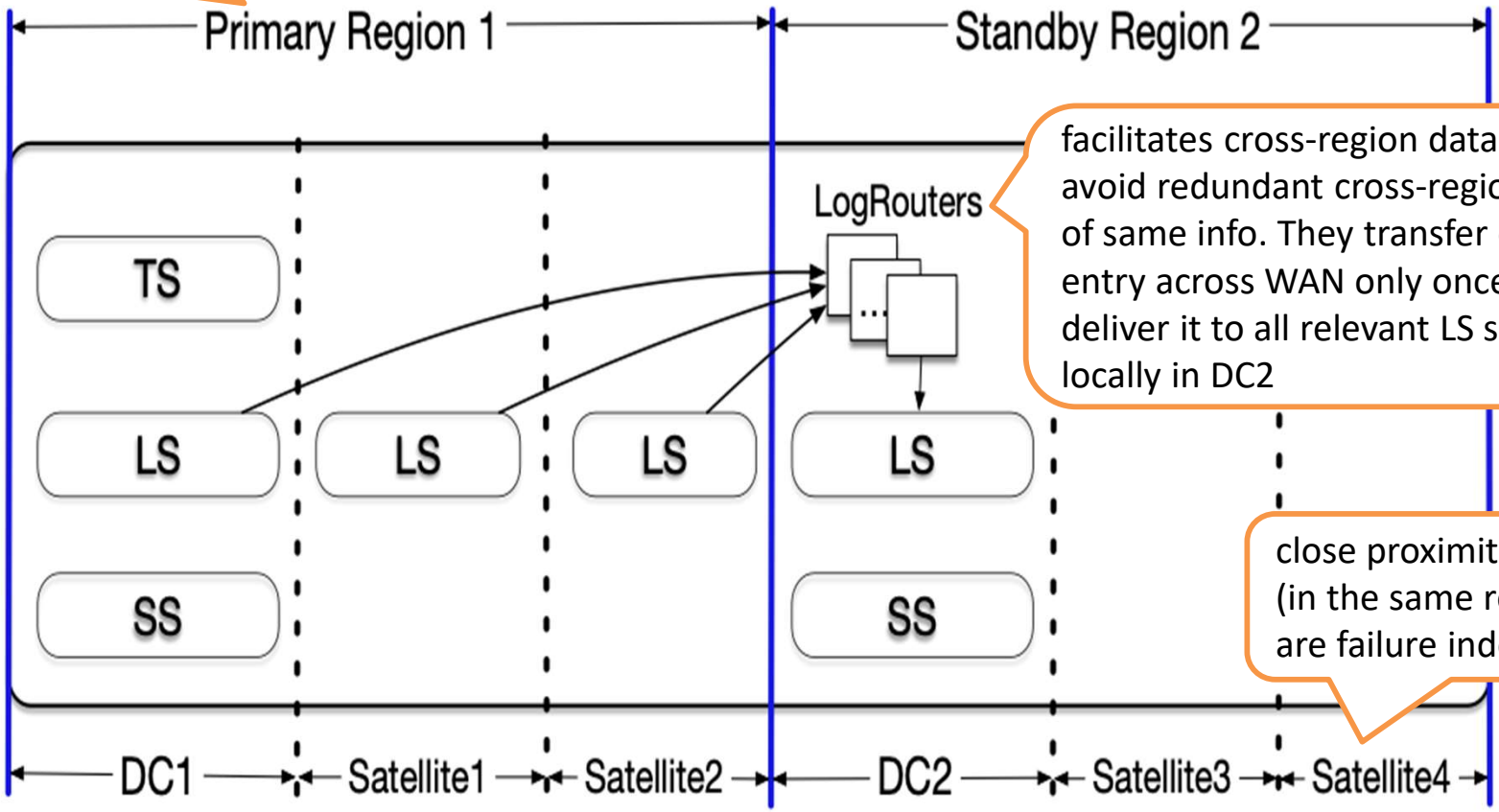
- main challenge of providing high availability during region failures is the trade-off of performance and consistency
  - Synchronous cross-region replication provides strong consistency but pays the cost of high latency.
  - asynchronous replication reduces latency by only persisting in the primary region but may lose data when performing a region failover
- FDB can be configured to provide Synchronous and Asynchronous replications
- Also, provides a high level of failure independence by leveraging multiple availability zones within the same region

Geo-replication Design

1. always avoids cross-region write latencies
2. provides full transaction durability, (like synchronous replication) so long as there is no simultaneous failure of multiple availability zones in a region
3. can do rapid and completely automatic failover between regions,
4. can be manually failed-over with the same guarantees as asynchronous replication (providing ACI, but not D of ACID) in the unlikely case of a simultaneous total region failure
5. only requires full replicas of the database in the primary and secondary regions' main availability zones, not multiple replicas per region

All client writes are forwarded to this region and processed by Proxies in DC1, synchronously persisted onto LSs in DC1 and 1+ satellites. Updates are then asynchronously replicated to DC2

Reads can be served from storage replicas at both primary and secondary data centers (consistent reads do require obtaining a read version from the primary)



facilitates cross-region data transfer, avoid redundant cross-region transfers of same info. They transfer each log entry across WAN only once & then deliver it to all relevant LS servers locally in DC2

close proximity to the DC (in the same region) but are failure independent

A two-region replication setup for an FDB cluster. Both regions have a data center and two satellite sites.

Coordinators are deployed across three or more failure domains (in some deployments utilizing an additional region), usually with at least 9 replicas.

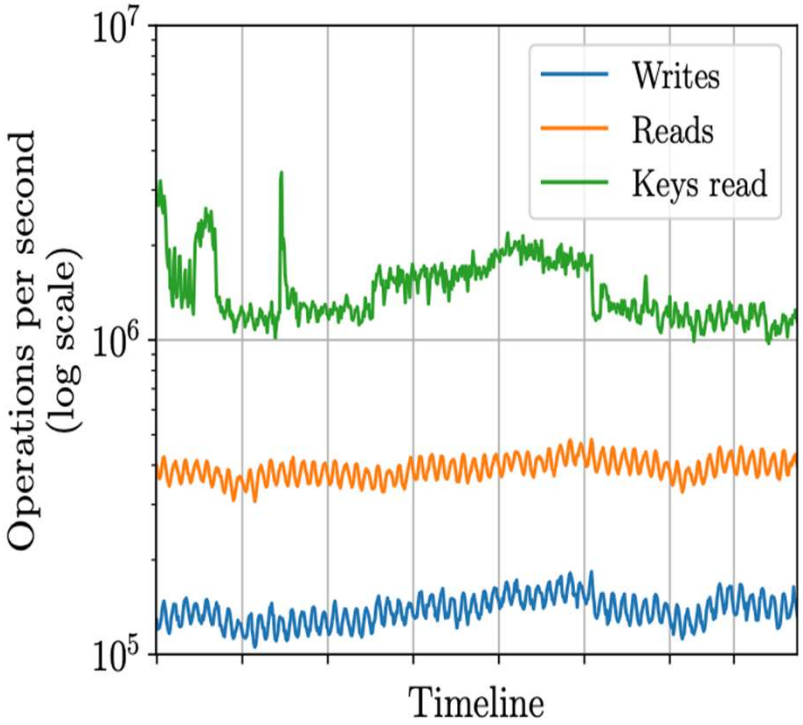
Primary DC Failure Recovery

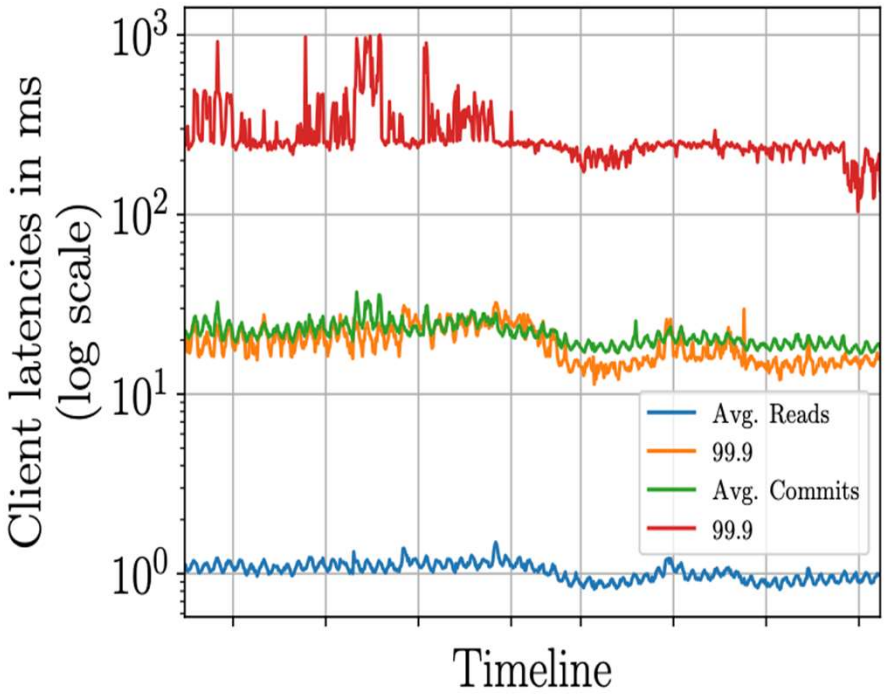
- The cluster (with the help of Coordinators) detects failure of Primary DC and starts a new TMS in DC2
- New LSs are recruited from satellites in secondary region as per the region's replication policy.
- During recovery, LRs in DC2 may need to fetch the last few seconds' data from primary satellites
  - They may not have made it to DC2 prior to the failover due to the async replication
- After the recovery, if the failures in Region 1 are healed and its replication policy can again be met, the cluster will automatically fail-back to have DC1 as the primary DC
  - due to its higher priority.
- Alternatively, a different secondary region can be recruited.

Production Measurement

- Measurement was taken from one of Apple's production geo-replicated cluster with 58 machines:
  - both the primary DC and remote DC have 25 machines (2 satellites in PR with 4 machines each)
- cluster is configured to use 1 satellite (6.5 ms latency) for storing log data and utilize the other (65.2 ms latency) running Coordinators.
- there are 862 FDB processes running on these machines, with additional 55 spare processes reserved for emergency usage.
- An SSD disk is bound to either 1 LS or 2 SSs processes to maximize I/O utilization
- measured client read operations at SSs and write (or commit) operations at Proxies.

Measurement from a production cluster for a month (hourly plots)

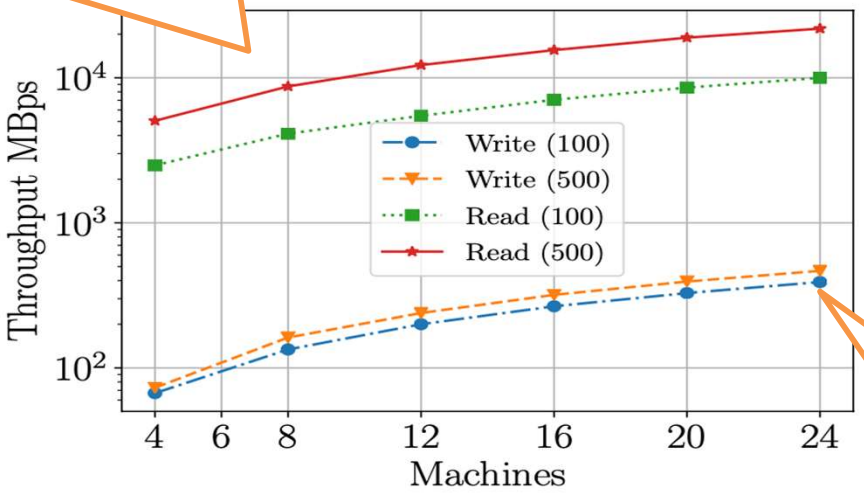




- For reads, 1ms for average and 19 ms for 99.9- percentile
- For commits, the average and 99.9-percentile are about 22 and 281 ms, respectively
- commit latencies > read latencies because commits always write to multiple disks in both primary DC and one satellite
- Recovery & Availability: 8/2020, one TS recovery ➔ 8.61 s

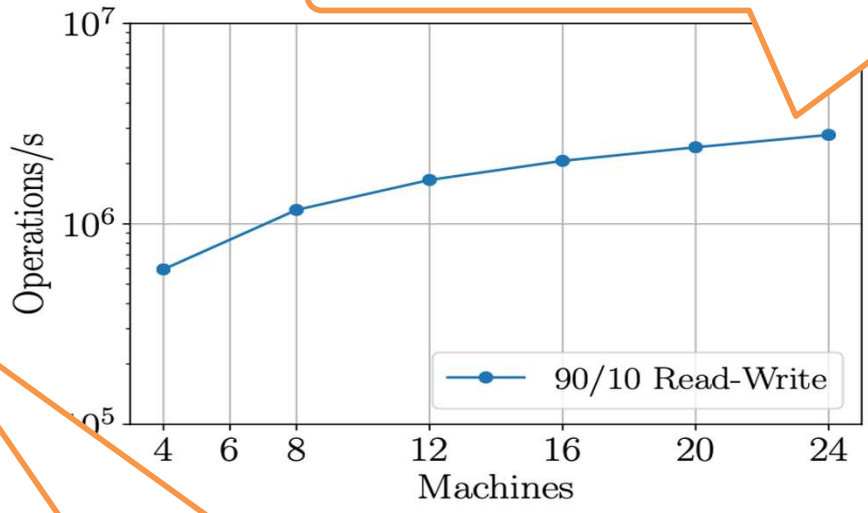
Scalability Test

Read throughput increases from 2,946 to 10,096 MBps (3.43X) for T100, and from 5055 to 21,830 MBps (4.32X) for T500



(a) Read and write throughput with 100 and 500 operations per transaction.

Ops increases from 593k to 2,779k (4.69X)



) 90/10 Read-write.

Writes throughput scales from 67 to 391 MBps (5.84X) for 100ops/trxn, and from 73 to 467 MBps (6.40X) for 500 ops/trxn.

synthetic workload with four types of trxns:

1) blind writes that update a configured number of random keys
2) range reads that fetch a configured number of continuous keys starting at a random key
3) point reads that fetch 10 random keys
4) point writes that fetch 5 random keys and update another 5 random keys.

blind writes used to evaluate the write and **range reads** for read performance

point reads and point writes are used together to evaluate the mixed read-write performance. (90/10 read-write) is constructed with 80% **point reads** and 20% **point writes** trxns

Reference

- *Jingyu Zhou, Meng Xu, Alexander Shraer, Bala Namasivayam, Alex Miller, Evan Tschannen, Steve Atherton, Andrew J. Beamon, Rusty Sears, John Leach, Dave Rosenthal, Xin Dong, Will Wilson, Ben Collins, David Scherer, Alec Grieser, Young Liu, Alvin Moore, Bhaskar Muppana, Xiaoge Su, and Vishesh Yadav. 2021. FoundationDB: A Distributed Unbundled Transactional Key Value Store. Proceedings of the 2021 International Conference on Management of Data. Association for Computing Machinery, New York, NY, USA, 2653–2666.*