UNIVERSITY OF CALIFORNIA
SANTA CRUZ

EdgeLab

Faisal Nawab
Natasha Mittal
Holly Casaletto
Abhishek Singh

https://edge-lab.github.io/
Byzantine faults . Data indexing .
Consensus algorithms . Data management
Coordinated Peer-2-Peer . Security .
Cloud application and infrastructure design
Fast distributed transaction processing
Distributed databases . IoT

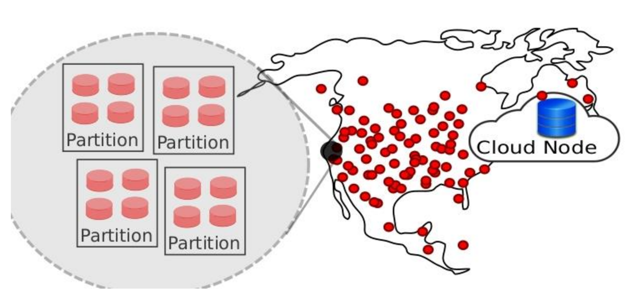
At UCSC EdgeLab, we study and build distributed data management systems. We are interested in systems that span large geographic areas and infrastructures, including cloud and edge environments.

Current projects:
* WedgeDB – Wide-area Edge Database
* LSM
* MinPaxos – Minority consensus

Edge databases : WedgeDB

System model 1

Motivation
- Avoid cloud overhead in transaction processing
- Locality-aware transaction processing

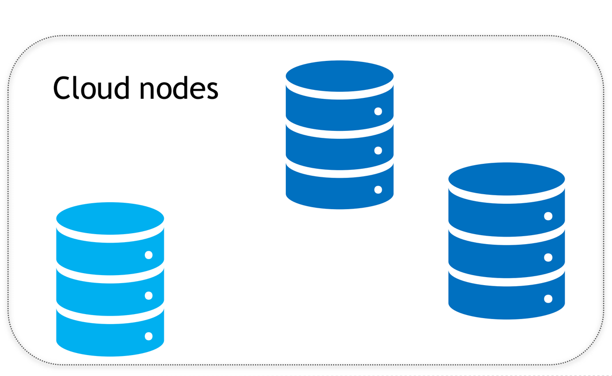


Description

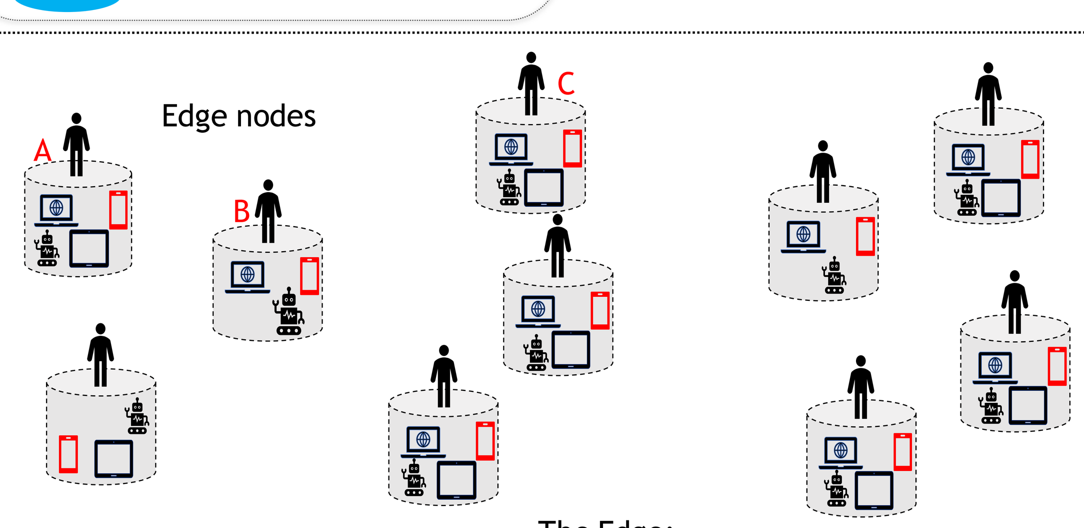
- Data is partitioned into clusters
- Partitions hold disjoint data
- Each cluster has a leader which coordinates transaction processing
- Transactions are batched and executed deterministically
- Every execution of a transaction batch builds a Merkle tree which is used as a proof for transaction execution.
- Efficient read-only transactions. Read-only transactions do not require consensus among nodes.
- Read-only transaction response includes verifiable proof of transaction other nodes involved in transaction

System model 2

Motivation
- Build cloud applications which do not access or manage user data directly.
- Users need never share personal data with anyone unless explicitly trusted.



The Cloud:
Used for data backup
Data availability
Trusted servers
Data integrity
Fault tolerance
Edge registry
Access permission management

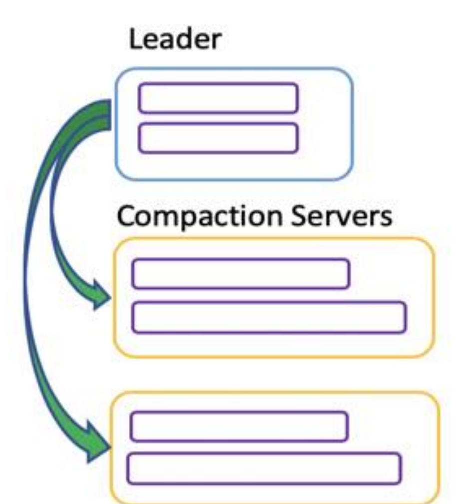


The Edge:
Primary data storage
Edge-Edge transactions
Edge-Cloud transactions

Description

- Data is held by users/edge nodes.
- Data encrypted by user's private key. Multiple public-private keys can be used for data encryption.
- Encrypted data is stored on cloud nodes. Cloud nodes do not have access to user data.
- Permissions are managed by cloud nodes. User B or C can request access to A's data via the cloud node by requesting for public keys to decrypt data.
- User A can agree to provide access to specific data by sharing the public key which can be used to decrypt A's data.
- Once permission is granted by A, encrypted data can be access from either the cloud node or directly from A. Access can be revoked by modifying public-private keys used to encrypt data and revoking permissions on the cloud nodes.
- This model allows cloud nodes to host application without accessing user's data without directly accessing user data.
- User has complete control of personal data.

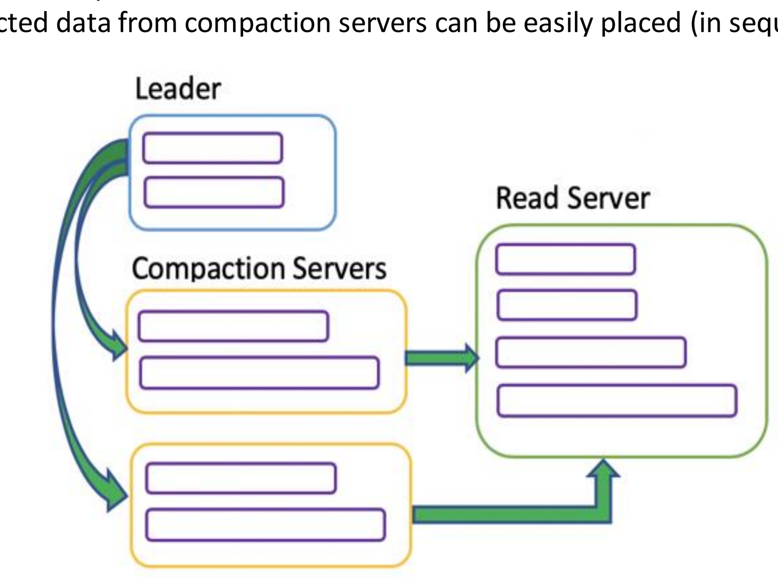
Cooperative Log-Structured Merge Tree (CooLSM)



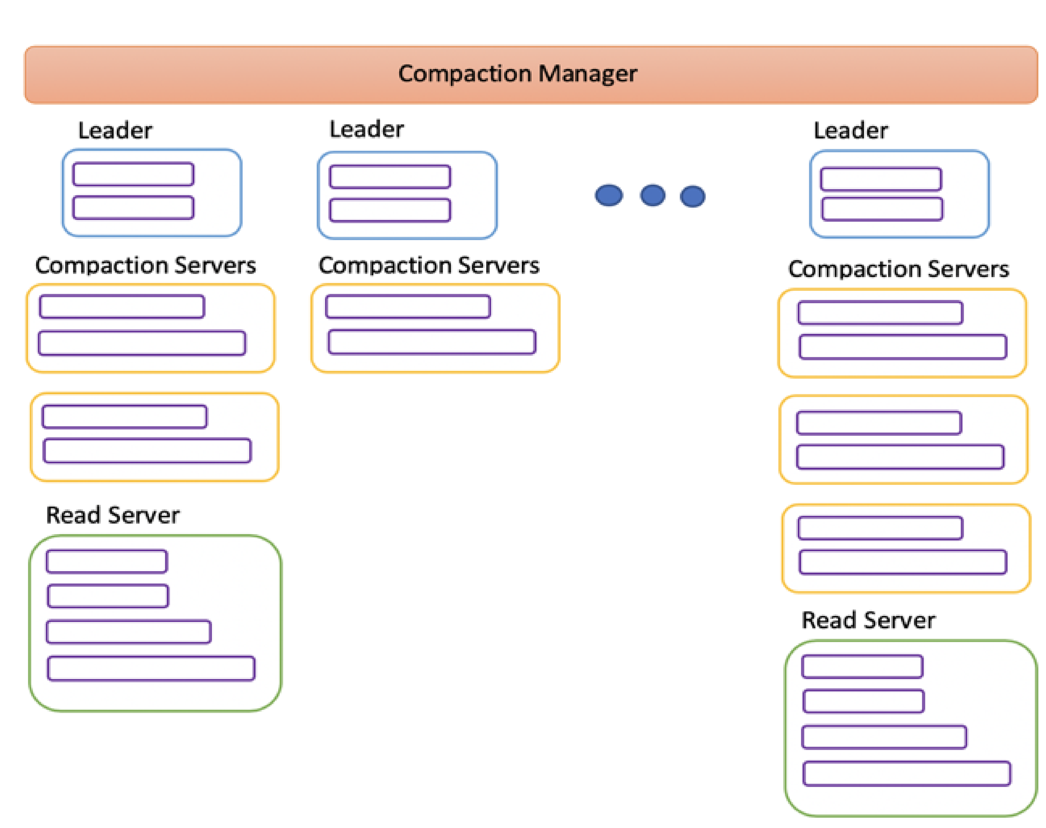
Compaction:
•maintenance operation
•negative impact on system resources
•degrades read performance

Dedicated Compaction Servers:
•eliminates CPU overhead from region servers
•faster compactions
•improves read performance

Read Server:
•improves read performance
•compacted data from compaction servers can be easily placed (in sequential batches)



CooLSM with Multiple Leaders



Motivation:
•Current LSM structure is monolithic which limits flexibility in terms of scalability.
•Only way to deal with increased load is to repartition data & distribute across nodes.

To do so, we break LSM tree into components, and then find a way to elastically scale these components.

Running more than one instance for each component can enable various performance advantages:
1.Increasing number of Leaders enable to digest data faster because we are no longer limited by performance of a single machine.
2.Increasing number of Compactors enable to offload compaction to more nodes and thus reduce impact of compaction on other functions.
3.Increasing number of read servers enable to increase read availability.

Minority consensus algorithms

MinPaxos and Sleepy Consensus

Motivation

- Traditional consensus models are not suitable for supporting the needs of emerging **IoT** and **edge applications**.
- IoT and edge applications are **unpredictable** and **sleepy**
  - Nodes can join and leave arbitrarily at any given time;
  - Nodes may voluntarily go to "sleep" to save energy as it sees appropriate;
  - Number of active nodes can be arbitrarily small compared to the total number of nodes at any given time.
- MinPaxos is **Minority Consensus** Model.
  - Can tolerate arbitrary number of failures;
  - Does not require votes from majority.
- MinPaxos trades consistency for **partition tolerance**. Partitioned minorities make progress while not hearing from each other.

Minority Consensus Guarantees

- **Validity**: Every block in the SMR log is one in which some user has requested to be committed.
- **At-most-once commitment**: A request to commit a block b cannot result in a SMR log with two copies of b.
- **Agreement**: There exists a time-difference function Δ such that the probability of two nodes disagreeing on the content of a position in the log at time (now - $\Delta(\varepsilon)$) is smaller than ε.
- **Termination**: There exists a time-difference function Δ such that the probability that at time (now + $\Delta(\varepsilon)$) a node will change its state of a committed block is smaller than ε.

Optimistic Leader Election

- Leader after sending **Prepare** message:
  - Checking if any objections
  - Waiting for threshold of time
  - Unilaterally self-proclaim

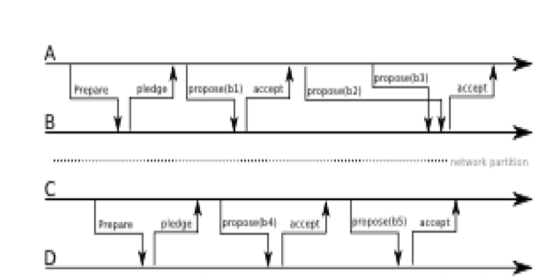


Figure 1: An example scenario of MinPaxos Leader Election and Replication.

Asynchronous Replication

- Replica after receiving a **Propose** message:
  - replicate the proposed block
  - request for blocks in the gap to be cont.

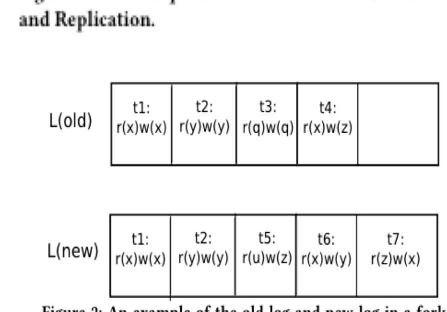


Figure 2: An example of the old log and new log in a fork merge scenario.

Resolving Forks

- MinPaxos sacrifices consistency for partition tolerance so forks are unavoidable.
- Borrows idea from Block Chain – Longest Chain Wins (**LGW**)

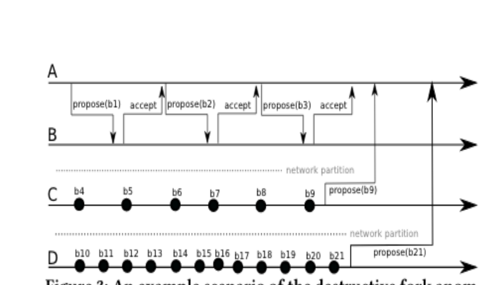


Figure 3: An example scenario of the destructive fork anomaly.

MinPaxos-TP

- Transactional merges increase concurrency;
- Re-committing transactions using MinPaxos Algorithm guarantees that the resulting log is serializable.

UC SANTA CRUZ

About us:
- Faisal Nawab | nawab.me
- Natasha Mittal | https://users.soe.ucsc.edu/~natasha/
- Holly Casaletto | https://users.soe.ucsc.edu/~hcasalet/
- Abhishek A Singh | twitter.com/alfredd

Email:
{fnawab, nmittal1, hcasalet, abasingh}@ucsc.edu