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
- 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.

**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 accessed 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)**

**Leader**
- Compaction:
  - Maintenance operation
  - Negative impact on system resources
  - Improves read performance

**Dedicated Compaction Server**
- Data compaction from region servers
- Faster compactions
- Improves read performance

**Compaction Servers**
- **Motivation**
  - Current LSM structure is monolithic which limits flexibility in terms of availability.
  - Only way to deal with increased load is to repartition data & distribute across nodes.

**Description**
- 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 digest data faster because we are no longer limited by performance of a single machine.
  2. Increasing number of Compactors enables 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**

**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 is seen appropriate
- Number of active nodes can be arbitrarily small compared to the total number of nodes at any given time

**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 - δ(t))/δ is smaller than e.
- **Termination**: There exists a time-difference function δ such that the probability that a new node at time (now + δ(t))/δ a node will change its state of committed block is smaller than e.

**Optimistic Leader Election**
- Leader after sending Prepare message
  - Checking if any objections
  - Waiting for threshold of time
  - Optionally, soft consensus.

**Asynchronous Replication**
- Replica after receiving a Propose message
  - Replica the proposed block
  - Request for blocks in the gap to be

**Resolving Forks**
- MinPaxos sacrifices consistency for partition tolerance so forks are unavoidable.
- Borrow idea from BlockChain – Longest Chain Wins (LAW)

**MinPaxos-TP**
- Transactional merges increase conciseness
- Re-committing transactions using MinPaxos Algorithm guarantees that the resulting log to serialize.

**About us:**
- Faisal Nawab
- Natasha Mittal
- Holly Casaletto
- Abhishek A Singh
- Natasha Mittal: https://users.soe.ucsc.edu/~nmittal1
- Holly Casaletto: https://users.soe.ucsc.edu/~hcasalet
- Abhishek A Singh: twitter.com/alfredd
- Faisal Nawab: {fnawab, nmittal1, hcasalet, abasingh}@ucsc.edu

**Email:** fnawab, nmittal1, hcasalet, abasingh@ucsc.edu