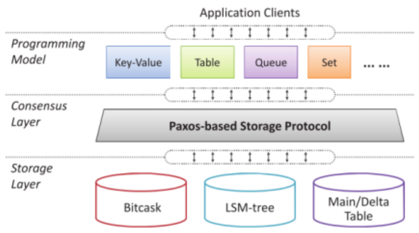
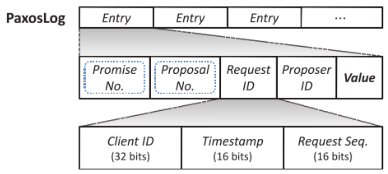
One Page Summary: PaxosStore: High-availability Storage Made Practical in WeChat

PaxosStore paper, published in VLDB 2017, describes the large scale, multi-datacenter storage system used in WeChat. As the name may suggest, it uses Paxos to provide storage consistency. The system claims to provide storage for many components of the WeChat application, with 1.5TB of traffic per day and tens of thousands of queries per second during the peak hours.

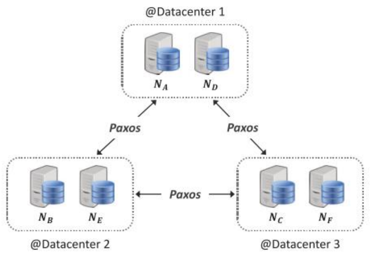


PaxosStore relies on Paxos protocol to for consistency and replication within tight geographical regions. The system was designed with a great separation of concerns in mind. At a high level, it has three distinct layers interacting with each other: API layer, consensus layer, and storage. Separating these out allowed PaxosStore provide most suitable APIs and storage for different tasks and application, while still having the same Paxos-backed consistency and replication.

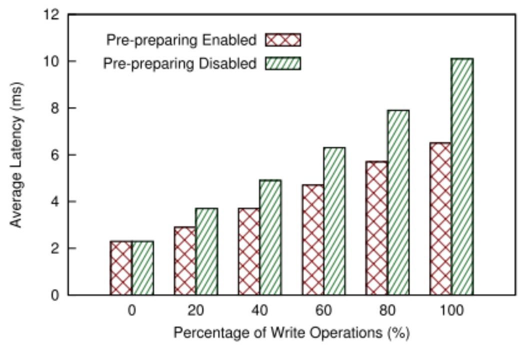


In a paxos-driven consensus layer, the system uses a per-object log to keep track of values and paxos-related metadata, such as promise (epoch) and proposal (slot) numbers. Log's implementation, however, seems to be somewhat decoupled from the core Paxos protocol. Paxos implementation is leaderless, meaning there are no single dedicated leader for each object, and every node can perform writes on any of the objects in the cluster by running prepare and accept phases of Paxos. Naturally, the system tries to perform (most) writes in one round trip instead of two by assuming some write locality. After the first successful write, a node can issue more writes with increasing proposal (slot) numbers. If some other node performs a write, it needs to have higher ballot, preventing the old master from doing quick writes. This is a rather common approach, used in many Paxos variants.

The lack of a single stable leader complicates the reads, since there is no authority that has the most up-to-date state of each object. One solution for reading is to use Paxos protocol directly, however this disrupts locality of write operations by hijacking the ballot to perform a read. Instead, PaxosStore resorts to reading directly from replicas by finding the most recent version of the data and making sure a majority of replicas have that version. This works well for read-heavy workloads, but in some high-contention (or failure?) cases the most-recent version may not have a majority of replicas, and the system falls-back to running Paxos for reading.



PaxosStore runs in multiple datacenters, but it is not a full-fledged geo-replicated system, as it only replicates between the datacenters located in the same geographical region. The paper is not clear on how data get assigned to regions and whether objects can migrate between regions in any way. Within each datacenter the system organizes nodes into mini-clusters, with each mini-cluster acting as a Paxos follower. When data is replicated between mini-clusters, only one (some?) nodes in each mini-cluster hold the data. This design improves fault tolerance: with a 2-node mini-cluster, failure of 1 node does not result in the failure of the entire mini-cluster Paxos-follower.



The paper somewhat lacks in its evaluation, but PaxosStore seems to handle its goal of multi-datacenter, same-region replication fairly well, achieving sub-10 ms writes.

This paper seems like a good solution for reliable and somewhat localized data-store. The authors do not address data sharding and migration between regions and focus only on the intra-region replication to multiple datacenters, which makes me thing PaxosStore is not really "global", geo-replicated database. The fault tolerance is backed by Paxos, mini-clusters and the usage of PaxosLog for data recovery. The evaluation could have been more complete if authors showed scalability limits of their system and provided some details about throughput and datacenter-locality of the workload in the latency experiments.