# On Brewing Fresh Espresso: LinkedIn's Distributed Data Serving Platform

Praveen Kamate / 22200326 / Group P

**Synopsis:**

LinkedIn's data ecosystem was quite simple in its early stages. As LinkedIn's customer base increased, the services and data requirements also grew, and RDBMS as a solution wasn't cost-effective for LinkedIn due to the high costs associated with hardware and caching. Also, the process of increasing the capacity requires a longer duration and was difficult to scale with zero downtime. Product agility presents a barrier for managing the schema, as a result, in 2009 Voldemort [2] was released. Based on the learnings from RDBMS and Voldemort, Espresso was designed.

Espresso is a timeline-consistent document-oriented distributed database. It provides a hierarchical data model to support the natural partitioning of data into collections that share a common partition key such as member id, etc. It follows the master-slave architecture, with the master node handling read and writes requests for a partition, and the slave reproduces the master so it can handle the master's failure. Due to the importance of low latency, the consistency is timeline consistent, with replication between master and slave being either synchronous or semi-synchronous depending on the application. In case of a failure of the master, the cluster manager elevates the slave to become the master after a configured timeout thereby satisfying the requirement of availability. It allows schema definition for the documents. Enforcing schemas allows systems across the entire data ecosystem to reason about the data in a consistent way and also enables key features such as secondary indexing, and partial updates to documents. It allows on-the-fly schema evolution i.e., it is possible to change the schema fields to a document and this change is backward compatible.

Espresso implements the local secondary indexing using Apache Lucene with index granularity per collection key. It also uses an indexing solution known as Prefix Index, each term in the index is indexed with the collection key for the documents, which reduces the latency and memory footprint. It partitions the data to address load balancing the requests and efficient cluster expansion. Each database partition functions as its own commit log. Each commit log serves as a history of data modification events on that partition, establishing an internal clock. These timelines are maintained in the data bus and help in case of node failures. It uses sharded MySQL with an enhanced binary log for replication and to guarantee consistency among replicas, the consistency checker is used. It compares the checksum of master and slave to detect any discrepancies. It uses Helix to discover fault tolerance, helix uses Zookeeper heartbeat for hardware failure, and also monitors performance metrics, in case of abnormal usage of resources, it considers as a failure. It also supports online cluster expansion and migrates partitions from existing nodes to new ones without pausing live traffic using Helix. Currently, LinkedIn serves majorly from a single data center and has a warm standby to assume charge in times of disaster.

The production releases of Espresso Storage Node, Data Bus Relay, Helix, and Router were evaluated. The total number of partitions was changed to study the impact on availability. As the number of partitions increases, group commit has a much lower failover delay than singular commit. But there is also an increase in the overall fail-over delay. Hence a very large number of partitions cannot be used. Secondly, the sharded MySQL was compared with Espresso, the Helix partition allocation was altered to make it mimic Sharded MySQL. In the fail-over latency for 64, 512, and 2048 partitions Espresso always outperforms the Sharded MySQL. Thirdly, the elasticity experiment showed that cluster expansion serially matches the expected percentage of expansion i.e., latency increases as the cluster expands. For cluster expansion in parallel, the higher the degree of parallelism, the faster the expansion. For the performance they consider small and large mailbox tests to compare the Lucene and Prefix index implementations. In the case of the small mailbox, the Prefix index outperforms Lucene by 2x because the Lucene index is immutable and for the large mailbox, the Prefix index outperforms Lucene by 2x because the large mailbox has a big index segment per mailbox. However, Lucene has better 99% latency than the Prefix index because mailboxes are big Prefix has a higher cost to modify the inverted index rows.

Espresso was built out of need and addressed trivial drawbacks in RDBMS and Voldemort with improved performance and secondary indexing. But the few areas to improve includes supporting cross-entity transactions that handle entity-entity edge relationships, multi-master deployment, improving the platform extensibility, and addressing the multi-tenancy challenges.

**References**

[1] On Brewing Fresh Espresso: LinkedIn's Distributed Data Serving Platform, SIGMOD '13

[2] Project Voldemort. http://project-voldemort.com/.