Wildfire: Fast HTAP on a loosely-coupled system
IBM Almaden Research Center and IBM Analytics*

Analytics over live, high-volume data streams is needed in domains ranging from finance (portfolio management) to healthcare to IoT. Traditional OLTP systems struggle with such workloads, both due to high ingest rates and due to the need for analytics on real-time data (called HTAP – hybrid transaction and analytic processing).

Another complication is loose coupling (“AP” in the sense of the CAP-theorem). Globalization has made ACID consistency pedantic: many modern applications, including conservative ones “where-money-lives”, want to do multi-master updates, with transactions committing even when the multiple masters are (transiently) disconnected. At the same time, these applications are not happy with eventual consistency. Wildfire is a prototype HTAP system being built to tackle these challenges. Wildfire simultaneously targets:

- HTAP with Open Format: Live transactional data ingested into Wildfire is continually groomed to low cost object storage (e.g., Cleversafe, S3), with full versioning, as Parquet blocks. Analytic queries submitted via Apache Spark can access transaction-consistent snapshots by directly accessing the object storage. Wildfire also provides a custom Spark API that allows Spark queries to see latest commits (snapshot isolation), by peeking into the log on Wildfire OLTP nodes.

- Versioning, Indexing, and Organizing:
  All modifications in Wildfire are upserts. But upserts are tricky on object stores that rely on immutable objects. For example, if a record inserted three years back is updated, how do we even detect this at commit, and can we hope to set an “end time” on the age-old version?
  A related problem is maintenance of indexes. Both primary and secondary indexes continue to be important for modern workloads: e.g., IoT applications often compute aggregates on all readings from a sensor within a time-range. But index maintenance has always been problematic with high update rates: index maintenance slows down commits; updates to even one field churns many secondary indexes; and so on.
  Wildfire solves both these issues by having a simple commit procedure, and an elaborate post-commit procedure, consisting of grooming and repeated rounds of post-grooming. The philosophy is that the tail portion of the data stays disorganized, to allow fast OLTP, and background operations merge this tail with more organized prior portions, in a hierarchical fashion.

- Multi-master, with AP: By default, transactions commit locally, and thus high ingest rates can be sustained even on poorly connected nodes. Queries however always perform quorum reads, and hence applications see monotonically advancing snapshots of the database state. But some high-value transactions need to be ACID, in two respects: (a) every subsequent query must see a committed high-value transaction, and (b) conflicting high-value transactions must be caught before commit.
  Wildfire transactions (and queries) can get this stronger consistency by requesting a specialized quorum that includes an elected shard-master.

- High-volume Ingests and Fast Point Queries: Wildfire targets an order-of-magnitude speed up in OLTP rates, and aims for 1 million ingests (upserts) per second per node, and many million point lookups per second per node. Fast ingests are achieved via careful engineering, asynchronous replication, and deferred version and index maintenance. For fast lookups, Wildfire uses a novel NVMe-optimized index based on sorted projections. Architecturally, Wildfire separates OLTP, Indexing, and Analytics onto separate nodes that can be scaled-out independently.

- Full serializability: Long-term, Wildfire aims for full serializability, without losing the multi-master or deferred maintenance characteristic, using novel symbolic transaction-execution methods.