Apache Kudu: Fast Analytics on Fast Data

Adar Lieber-Dembo

Why should you care?

Kudu is a scale up, scale out, columnar, open source storage engine

- Exploits modern hardware (e.g. NVMe, gobs of RAM)
- Sharded share-nothing design
- Column-oriented disk layout: efficient storage and fast analytical scans
- Structured schema: familiarity and ergonomics
- Supports many query engines, such as Apache Impala and Apache Spark
- Code licensed under ASL 2.0 and active project community

Fast Analytics on Fast Data

For data sets that are:

- Constantly growing and you want to query the latest data ASAP
- Updated
- Used in both analytical scans and point lookups
- Structured and have evolving schemas

Kudu provides:

- Immediate availability (for scans) of rows as they are written
- Seamless updates and deletes

Schema

- Finite number of columns
- Each column has a type, encoding, compression, default value, and nullability
- Primary key is a subset of all columns
  - Defines row uniqueness
  - Defines sort order of rows on disk
  - Basis for clustered index: profound effect on performance
- Partitioning: range, hash, or both
  - Each level defines its own partition key (must be a subset of primary key columns)
  - Range partitions can be added or removed; hash buckets are fixed (for now!)

Clients

- Clients write data into Kudu
  - Data organized into rows
  - Each row operation may be one of INSERT, UPDATE, DELETE, or UPSERT
  - Servers enforce primary key uniqueness/existence
- Clients scan data out of Kudu
  - Scan may include projection of columns
  - Scan may include predicates on arbitrary columns
- Speaking of predicates...
  - Predicated on a prefix of the primary key? Will use clustered index
  - What about on a prefix of a partition key? Entire partitions can be pruned
  - Predicates are pushed down to per-column scanners
  - Column values are encoded so we can evaluate predicates without decoding data

Replication

- Partitions are logically replicated via Raft consensus protocol
- Writes must be sent to leader
  - Committed when on majority of WALs
- Scans may go to any node
  - No quorum necessary for scan
- Table/partition/schema catalog is a Raft-replicated single partition

Things I think are cool

HybridTime (aka Hybrid Logical Clocks)

- Each timestamp has a physical and logical component
  - Both may only move forwards
- HT timestamp assigned to each write
- Timestamps exchanged between servers and between clients and servers
  - Keeps time disparities low across the cluster
- Fully linearizable in the absence of hidden channels
  - If hidden channels exist, must propagate timestamps across them for linearizability

Ergonomic alternative to Lamport Clocks (i.e. can scan at a particular point-in-physical-time which is also a HT timestamp)

Things I think are cool

MVCC

- Implemented via full fidelity history retention
- Every operation has a HT timestamp
  - Maintained faithfully during all flushes and compactions
- All scans have snapshot isolation consistency
  - Never read “dirty” data
- May also scan between two points-in-time to produce “difference”
  - Used in incremental backups

Things I think are cool

Storage is a veritable fractal of LSM trees

- One LSM tree for “base” data
  - Large in-memory store flushes to new disk components each with min/max PK bounds
  - Disk components periodically compacted together to reduce PK overlap
  - Compaction is budgeted and light-weight to avoid hogging CPU/IO for a long time
- An LSM tree for mutations in each disk component
  - Mutations bypass main in-memory store in favor of per-disk-component memory store
  - Flushed into their own disk components, periodically compacted together
  - Periodically compacted into base data to reduce number of mutations applied in a scan
  - Post-compaction, mutations are kept in an inverted state to enable “time travel” scans
- Separating base data and mutations: pure INSERT write performance can be comparable to that of an immutable storage engine

Things I think are cool

Disk components

- Each one has min/max PK bounds (stored in-memory)
  - Builds shard-level interval tree to prune components during scan and when checking for existence of a PK during write
- Each one has a bloom filter (paged from disk)
  - Also used to prune components when checking for existence of a PK during write
- Each one has a PK B-tree index (paged from disk)
  - Great for point lookup “scans”
- Mutations based on Positional Delta Tree design
  - Mutation key is {row ordinal, timestamp} rather than full PK
  - Much cheaper to apply on top of base data at scan time

Things I think are cool

Memory stores

- Based on MassTree design
  - No record removal; deleted records removed at flush time
  - No in-place record updates; atomic CAS to append mutations to a per-record linked list
    - Taken together: MVCC on records
  - Leaf nodes are linked via next pointer (as in B+-tree): improved sequential scan performance
  - No full “trie of trees”: less concerned about extremely high random access throughput
- SSE2 memory prefetch instructions to prefetch one leaf node ahead of scan
- JIT-compile record projection operations using LLVM

Things I think are cool

New (maybe?) merge algorithm

- Kudu ordered scans
  - Input: N lists of rows sorted by PK, one from each component
  - Output: one mega list of rows sorted by PK
- New algorithm improves upon traditional heap-based merge
  - Minimizes size of heap (and thus heap motion) using state tracked in two additional heaps
- Can be used in compactions in the future
  - Usable in any domain with limited “peek ahead” of list elements

More information

- http://kudu.apache.org/
- http://github.com/apache/kudu
- http://kudu.apache.org/kudu.pdf
- http://users.ece.utexas.edu/~garg/pdslab/david/hybrid-time-tech-report-01.pdf
- https://getkudu-slack.herokuapp.com/