Understanding the Science of Databases

June 23rd, 2015 by Thomas Hazel

The State of the Art of information storage is based on two fundamental algorithms: B-Treeand LSM-Tree, where many LSM implementations actually use B-Tree(s) internally. In other words, over the last 30 years these algorithms and their corresponding data structures have been the cornerstone to just about every Row-Oriented, Column-Oriented, Document-Oriented, and even File-System architecture.

Though there are variants to either of these designs, both have specific behavior to handle particular use-cases. For instance, B-Tree(s) are typically seen as "read-optimized" algorithms and LSM-Tree(s) are seen as "write-optimized". Each has a Big-O-Notation describing the cost to Create, Read, Update, and Delete (CRUD) information, where random (i.e. unordered) information is more costly to operate on than sequential (i.e. ordered) information. In the database world, cost is most associated to information manipulation on physical storage. So limiting such cost (e.g. seek time or write amplification) is key to improving performance.

To get around the inverse limitations of these algorithms and reduce their overall cost, architectures have implemented pre and post "workarounds". For example, Row-Oriented architectures (e.g. RDBMS) have introduced Log-File (WAL) in front of the underlying B-Tree so that information can be pre-organized to limit the cost of writes (i.e. writes requiring reads). On the contrary, LSM architectures typically use blind writes (i.e. writes not requiring reads) and post-organize via log merge leveling to limit the cost of subsequent reads.

Though their mileage may vary, each implementation has distinct performance metrics. On average B-Tree(s) are typically 2x faster than LSM-Tree(s) on reads and LSM-Tree(s) are typically 2x faster than B-Tree(s) on writes. Yet what is sometimes missed in such metrics is where the testing is done. For instance, MySQL (Row-Oriented) has requirements (e.g. ACID) that put a greater burden on the underlying B-Tree than let's say MongoDB's (Document-Oriented) underlying B-Tree. And comparing MySQL to let's say Cassandra (Column-Oriented) is even more problematic because not only are the requirements different so are the underlying algorithms (B-Tree vs. LSM-Tree respectively).

Therefore if MySQL could out perform Cassandra on a write heavy use-case or vice versa Cassandra out perform MySQL on a read heavy use-case, it would say a lot about the power of the underlying algorithm. And typically the underlying algorithm "hits the wall" due to their degenerate use-case and the performance of physical storage so if an algorithm could

limit reads (seek) and/or writes (amplification) to their theoretical minimum, it would be a big leap over the State of the Art of information storage.

Achieving theoretical minimum cost and thus obtaining maximum performance in the abstract is quite simple. In other words, the performance of writes (fully acid) should have a seek cost of “0” grouped for peak throughput. Performance of reads (point or scan) on the other hand should have seek cost of “1” where query column and/or row selection is precisely optimized. And finally, storage utilization should include maximizing both processor and memory resources as well. Achieving theoretical minimums in the applied is where the reimaging begins.