Tokutek®

TokuDB®: Scalable High Performance for MySQL® and MariaDB® Databases

Technology Whitepaper

April 2013

"The real end-game for Big Data is to have transactional and analytic data on the same database." -- David Floyer, Wikibon

Overview

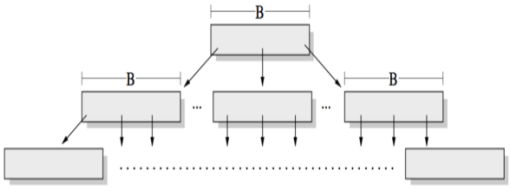
TokuDB for MySQL® & MariaDB® is a high-performance storage engine that increases MySQL & MariaDB performance and scalability on mixed workloads by one to two orders of magnitude. Rather than optimizing for a narrow or specialized set of use cases by re-engineering existing technology, Tokutek introduces a new Fractal Tree® indexing algorithm, which is based on Cache-Oblivious Algorithmics, a fundamentally new approach to building memory-efficient systems that was pioneered by Tokutek.

This whitepaper will:

- Give an overview of B-trees, the indexing method used by today's conventional databases
- Outline the performance characteristics of Fractal Tree Indexes and TokuDB
- Describe Tokutek's advantages in three main areas: Performance, Agility, and Compression

Brief Database History

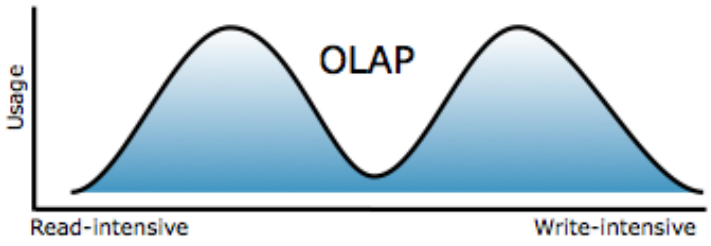
A database is a collection of data organized in such a way that a computer program can quickly select desired pieces of data.



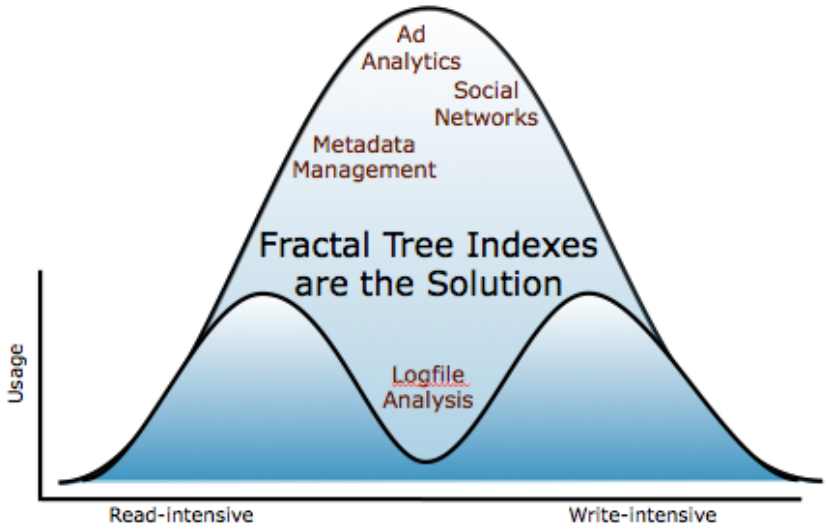
B-Tree

For the last 40 years, almost all existing databases index data using the B-tree, a data structure invented by Rudolf Bayer and Ed McCreight while working at Boeing Research Labs. The B-tree was developed to store huge amounts of data for fast retrieval on disk drives. The performance of the B-tree worked fine with the slow disks from 1970's era hardware.

While they were adequate for sequential workloads, B-trees fail for more information rich (random) workloads. With the progress in technology in the areas of processors, computer memory, computer storage, and computer networks, the sizes, capabilities, and performance of databases have grown by orders of magnitude. To keep up, over the last two decades, specialty solutions around online analytical processing (OLAP) and online transaction processing (OLTP) databases were developed as compromises to get around dated, rigid indexing technology.



Today's databases require smarter software algorithms that keep pace with hardware trends. They require new technology for today's denser drives and faster flash technology. Tokutek has developed a new data structure, Fractal Tree Indexes (FTIs), that perform up to two orders of magnitude faster than B-trees while requiring less tuning and administration. The user benefits of FTIs include: unmatched speed, high compression, and exceptional agility.



Fractal Tree Indexes Technology Overview

Indexing is at the foundation of databases. Build a better foundation, and the whole structure improves. TokuDB indexes data using Fractal Tree indexes, a new approach to indexing invented by Tokutek. Based on a class of algorithms first introduced in 1999, Fractal Tree Indexes (FTIs) are designed around the way that hardware has changed since the B-tree was invented: multi-core processors and high-bandwidth storage systems. They outperform B-trees on hard disks (HDs) and on solid-state drives (SSDs), and they will continue to outpace B-trees with every hardware generation.

> "We don't know how we could have gotten to our required scale and price points for our meta-directory components without TokuDB."
>
> -- Limelight Networks

This section of the paper explains why Fractal Tree Indexing is superior to B-tree indexing on both HDs and SSDs. Although the advantages of TokuDB over a B-tree-based solution like InnoDB seem to be disparate and varied, they are based on two related ideas: message propagation and I/O buffering.

In MySQL and MariaDB, the indexing algorithm can be upgraded simply by dropping in a new storage engine since they both support a well-defined storage engine API. It is the job of the storage engine to keep track of a set of items for insertions, deletions and updates, while supporting point and range queries. The task is challenging since large data -- data that is too big to fit in memory -- needs to be stored on disk.

Message Propagation

In a traditional B-tree index, when an update is made to the index, the index is changed to reflect this update. When a row is added, the leaf where the row will end up is modified to add the row. When a row is deleted, the leaf where the row resides is immediately changed to eliminate the row.

In an FTI, each change is a message. That message makes its way to the leaf, but the leaf itself is not modified until the message gets there. A message gets injected into the root of the FTI and it may stay there for some time. As long as the query

takes the same path as any message affecting the query, the query will reflect all relevant changes, even those that have not made it to a leaf.

Consider the most dramatic example: column addition or deletion. In a traditional B-tree, it is quite tricky to add or remove a column without locking the table. In InnoDB, this has yet to be achieved. Solutions that change an InnoDB schema either lock the table for a considerable time or rebuild the table in the background, so that an added column is not available for some indeterminate amount of time.

In TokuDB, an add column or delete column message can be added to the root virtually instantly. Once there, it is broadcast to all leaves, and the broadcast message makes its way down to the leaves just like any other message. The significance to the user is that a column addition or deletion is almost instantaneous and does not lock the table. Leaves are rewritten with the new schema only in the course of normal operation when they would be fetched into memory anyway, but the schema change takes immediate effect.

Messaging is a powerful tool in databases. Consider the problem of keeping track of counters on a busy website. InnoDB would need to fetch in leaves to update the counters. TokuDB simply injects a message at the root that updates the counter. The difference can be orders of magnitude faster.

I/O Buffering

Every Input/Output (I/O) to disk takes many orders of magnitude more time than an I/O to main memory and often limits database performance. Hard disks can only perform 200 or so I/Os per second (IOPS), which can easily become the bottleneck of a database system. Even SSDs don't fully address the problem. Although they perform many more IOPS than hard disks, inefficient I/O can use up a lot of bandwidth, and it is very costly to build a high-bandwidth storage system from SSDs. It is therefore critical to database performance to make the most of each I/O.

The best way to improve the performance of I/Os is to buffer updates. When updates are buffered, they are saved up so that a group of them can be written during a single I/O operation. This is not a new idea: buffering approaches to improving I/O are as old as databases.

The difference with TokuDB is that FTIs use provably optimal buffering. That is, it is possible to how that there is a mathematical minimum amount of I/Os to build and maintain an index, and FTIs match this provable lower bound. The I/O shortcomings of B-trees can be somewhat mitigated by buffering, for example by the InnoDB insertion buffer. These allow B-trees to improve their performance on certain workloads. However, FTIs incorporate buffering throughout the index, thereby reducing I/Os and improving bandwidth efficiency for every workload.

Consider the following analogy. Suppose a national distributor had a single point of distribution but receives orders from all over the country. Suppose that every order was delivered separately, that is, each order would be placed on a truck by itself for delivery. The fuel costs of delivery would be very high. This is what unbuffered B-trees do.

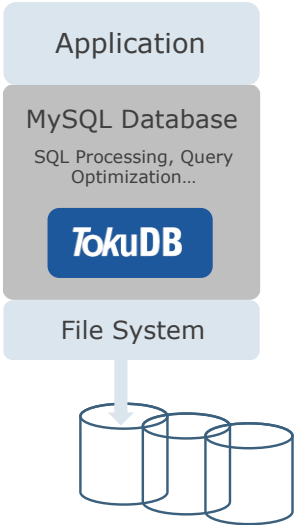
A B-tree with an insertion buffer, like InnoDB, keeps a single warehouse. Items accumulate for delivery in the warehouse, and when the warehouse is full, the truck is loaded with some item. If more than one item is going to the same zip code, then those items are all loaded and delivered. The per-item cost of delivery goes down, but only by the average load on the truck -- say two to four in a typical database application. The bigger the warehouse, the luckier we get with repeated zip codes, but for the scheme to work for a B-tree, the "warehouse" can be no bigger than RAM.

FTIs use a scheme that is analogous to a network of warehouses. Consider a system by which there are regional warehouses, then statewide warehouses, and finally local warehouses. The truck from the central warehouse gets filled with items going to the same regional warehouse, and when there are enough items going to the same statewide warehouse, they get loaded onto a truck. The cost of each trip is split among a large number of items.

Back in the world of databases, the number of I/Os per insertion drops, the bandwidth efficiency increases, and the speed of updating indexes increases. B-trees tend to update very little of a leaf at a time, because of their relatively poor use of buffering. Thus, leaves tend to be small in most implementations. In order to see why, consider a leaf that holds 100 rows, where one row is modified. When this leaf gets written, 99% of the bandwidth is wasted because 99 of the 100 rows were read and now written without modification. If the leaves were bigger –

suppose they held 1000 rows – and one row were modified, then 99.9% of the bandwidth would be wasted. Thus, B-trees tend to have small leaves. Also, they do a poor job of compressing because compression algorithms do better when they can compression a bigger chunk of data. FTIs can keep bigger leaves because they buffer well and update much of a leaf whenever they update a leaf so they use the bandwidth more efficiently. Additionally, an important side effect of large leaves is high compression.

Key Benefits of TokuDB for MySQL and MariaDB



TokuDB is designed to be a drop-in replacement storage engine for MySQL and MariaDB. A software-only plug-in, TokuDB is ACID- and MVCC-compliant and is fully compatible with existing MySQL and MariaDB applications. It requires no modifications to existing code or application logic.

ACID (atomicity, consistency, isolation, durability) Compliancy refers to a set of properties that insure reliable database transactions in these ways:

- Atomicity: applies the principle of the atom (the smallest indivisible particle) to database transactions so that the queries that make up the database transaction must either all be carried out or not.
- Consistency: refers to the rules of the data and keeping them consistent throughout the transaction.
- Isolation: ensures that data being used for one transaction cannot be used by another transaction until the first transaction is complete
- Durability: once a transaction has completed, its effects should remain and become irreversible

Multi-Version Concurrency Control (MVCC) Compliancy refers to a technique for improving multi-user database performance. It does this by eliminating row-level locking and table locking. This ensures that locks acquired for querying (reading)

data will not conflict with locks acquired for writing data and so reading never blocks writing and writing never blocks reading.

Performance, Agility, and Compression

Tokutek offers advantages in three main areas: Performance, Agility, and Compression.

Performance

With a 10x or more improvement in insertions and indexing, TokuDB delivers faster, more complex ad hoc queries in live production systems without the need to rewrite or tune an application. Offering high performance even when tables are too large for memory, TokuDB scales MySQL and MariaDB far beyond either InnoDB or MyISAM.

> "For us, TokuDB proved to be over 50x faster to add and update data into big tables. Adding 1M records took 51 minutes for MyISAM, but 1 minute for TokuDB."
>
> -- University of Montreal Genomics Laboratory

Tokutek developed a popular open source benchmark test called iiBench that measures how fast a storage engine can insert rows while maintaining secondary indexes. This is often a critical performance measurement since maintaining the right indexes will dramatically improve query performance.

Utilizing the iiBench test on a database with one billion rows inserted into a table while maintaining three multi-column secondary indexes, the TokuDB Fractal Tree Indexes remained at a steady insertion rate of 17,028 inserts/second whereas InnoDB dropped to 1,050 inserts/second. That's a difference of over 16x.

Compression

By leveraging write-optimized compression, TokuDB achieves up to a 90% reduction in HDD and flash storage requirements, without impacting performance.

TokuDB compresses large blocks of data — on the order of MBs, rather than the 16KB blocks that InnoDB uses — that is a big part of why TokuDB offers better compression. In benchmark tests InnoDB compression proved much lower than TokuDB because it is forced to work with smaller block sizes. InnoDB compression is further hampered by the choice to maintain fixed-size blocks on disk.

> "The impact to our storage was dramatic. In our comparison benchmarks, we went from 452GB with InnoDB to 49GB with TokuDB"
>
> -- Southwest Research Institute

Agility

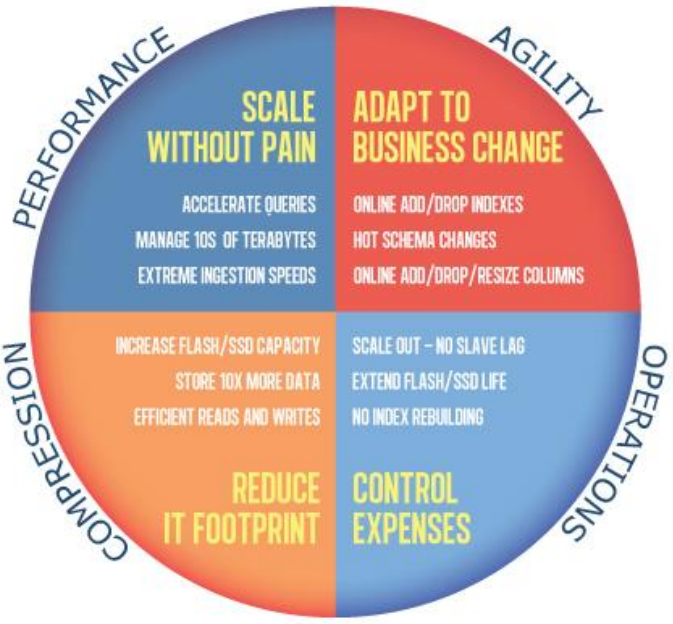
TokuDB offers agility at scale with Hot Schema Changes. Hot Column Addition/Deletion/Rename features allow for much larger tables to be created. It also provides the ability to add, drop, or rename a database column or field quickly and easily *without taking the database offline*.

Hot Indexing enables ad hoc queries to run fast with realtime, optimized index support. It allows for concurrent operations on the database and the index. Hot indexing also brings the familiar Enterprise Database online operations to MySQL and MariaDB.

> "Column additions in the past were simply not practical – taking days to complete. Now they can be done in a matter of seconds and accomplished in a non-disruptive fashion."
>
> -- Intent Media

Additional Benefits

Providing immunity to database aging by eliminating the need to rebuild indexes, TokuDB ensures no more query slowdowns, no more dump/reload, and no more need for dedicated maintenance windows. In addition, by utilizing larger, less frequent I/O, TokuDB reduces wear for flash thereby extending their service life. Finally, with high insertion rates, TokuDB addresses the common and persistent problem of "slave lag" in which a replication server is unable to keep up with the query load borne by the master server.



Conclusion

Customer experiences with TokuDB deployed in production continue to provide compelling evidence of the advantages of Fractal Tree™ based storage over conventional B-tree based storage. For a wide range of query types, storage sizes, insertion rates, and workload types that reflect the real world need to simultaneously store *and* query, Fractal Tree performance is 10x-50x faster than conventional B-tree based designs.

About Tokutek, Inc.

Tokutek is a performance engine company that delivers 21st-Century capabilities to the leading open source data management platforms. Tokutek applies patented Fractal Tree™ Indexing to increase MySQL performance and MongoDB performance, decrease database size and minimize downtime. As a result, Tokutek allows you to build a new class of applications that can handle unprecedented amounts of incoming data and scale to handle the data of tomorrow. The company is headquartered in Lexington, MA, and has offices in New York, NY. For more information, visit Tokutek.com or follow us on Twitter @Tokutek.

About MySQL

MySQL is the most popular open source database software in the world. Many of the world's largest and fastest-growing organizations use MySQL to save time and money powering their high-volume websites, critical business systems, communications networks, and commercial software. MySQL is owned by Oracle, the world's largest business software company. For more information, visit http://oracle.com.

About MariaDB

MariaDB strives to be the logical choice for database professionals looking for a robust, scalable, and reliable RDBMS (Relational Database Management System). MariaDB can be deployed as a drop-in replacement for the popular MySQL database and it is built by the original MySQL architects and most of the original core developers of MySQL with assistance from the broader community of Free and open source software developers. In addition to the core functionality of MySQL, MariaDB offers a rich set of feature enhancements including alternate storage engines, server optimizations, and security and performance patches. For more information on MariaDB visit http://mariadb.org and http://kb.askmonty.org.