Tokutek®

Optimizing MongoDB® With Fractal Tree® Indexes

2012 MongoDB Benchmark Summary

Optimizing MongoDB With Fractal Tree Indexes

Tokutek benchmarked standard MongoDB indexing against MongoDB with Tokutek's Fractal Tree® Indexes. The benchmarks measured the following scenarios:

- *Benchmark 1: indexed insertion on document with 4 secondary indexes*
- *Benchmark 2: indexed insertion on document with 4 secondary indexes, secondary index range query running once per minute retrieving 1000 documents*
- *Benchmark 3: indexed insertion on document with 4 secondary indexes, covered secondary index range query running once per minute retrieving 1000 documents*

BENCHMARK #1:
10x Insertion Performance Increase for MongoDB with Fractal Tree Indexes

The challenge of handling massive data processing workloads has spawned many new innovations and techniques in the database world, from indexing innovations like Tokutek's patented Fractal Tree® technology to a myriad of "NoSQL" solutions. Among the most popular and widely adopted NoSQL solutions is MongoDB. In order to find out if Fractal Tree indexing could offer some advantage when combined with MongoDB, Tokutek implemented a "version 2" IndexInterface as a Fractal Tree index and ran some benchmarks. Note that this integration only affects MongoDB's secondary indexes; primary indexes continue to rely on MongoDB's indexing code.

For the initial benchmark, Tokutek measured the performance of a single threaded insertion workload. The inserted documents contained the following: URI (character), name (character), origin (character), creation date (timestamp), and expiration date (timestamp). A total of four secondary indexes were created: URI, name, origin, and creation date. The point of the benchmark is to insert enough documents such that the indexes are larger than main memory and show the insertion performance from an empty database to one that is largely dependent on disk IO. The benchmark was run first with journaling disabled, then again with journaling enabled.

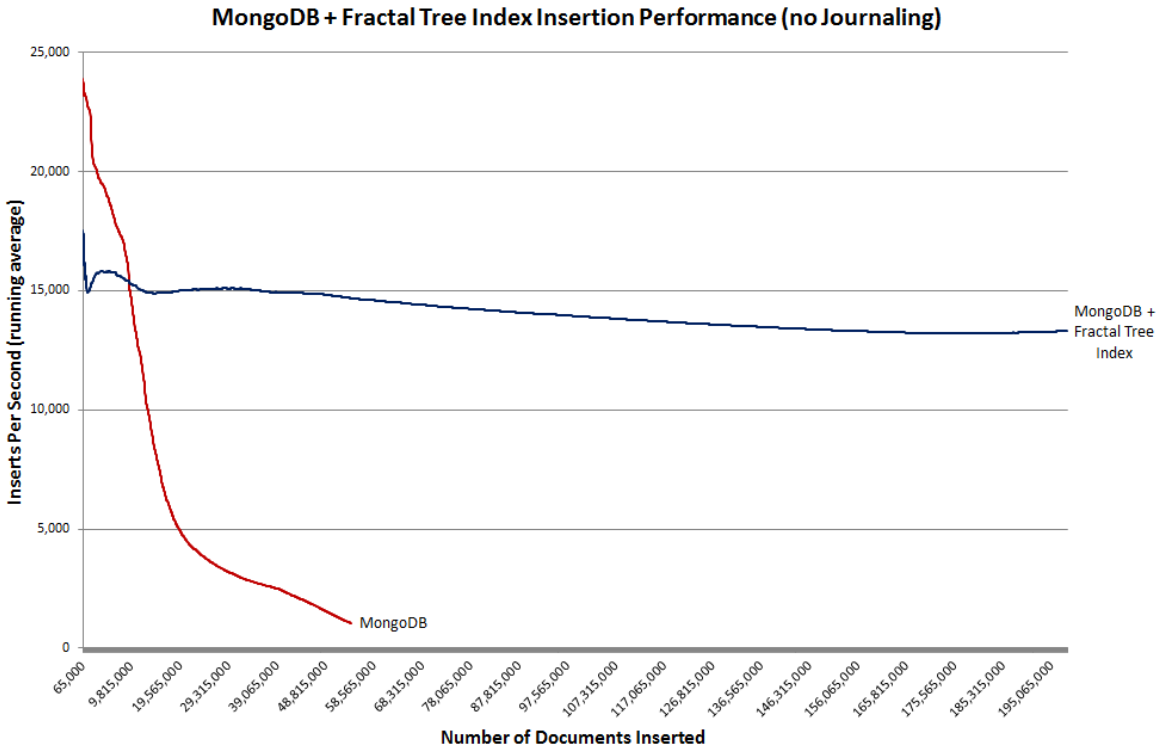
Benchmark Environment

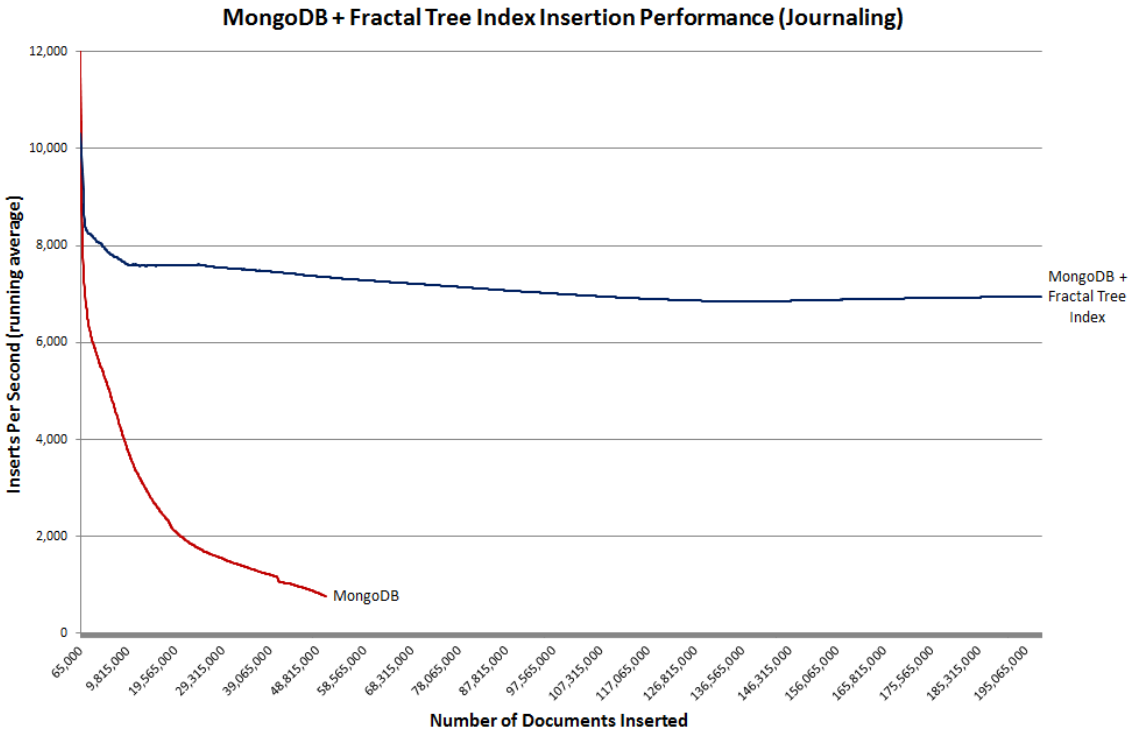
Sun x4150, (2) Xeon 5460, 8GB RAM, StorageTek Controller (256MB, write-back), 4x10K SAS/RAID 0

Ubuntu 10.04 Server (64-bit), ext4 file system

MongoDB v2.2.RC0

Benchmark Results





The exit velocity of standard MongoDB was 1,045 inserts per second at 54 million document insertions versus MongoDB with Fractal Tree Indexes exit velocity of 13,304 inserts per second at 198 million document insertions: an improvement of **1273%**. With journaling, MongoDB = 763 and MongoDB/FTI = 6951: an improvement of **911%**.

BENCHMARK #2:
268x Query Performance Increase for MongoDB with Fractal Tree Indexes

Tokutek continued its experimental integration of Fractal Tree® Indexes into MongoDB, adding support for clustered indexes (http://www.tokutek.com/2009/05/introducing_multiple_clustering_indexes). A clustered index stores all non-index fields as the "value" portion of the index, as opposed to a standard MongoDB index that stores a pointer to the document data. The benefit is that indexed lookups can immediately return any requested values instead of needing to do an additional lookup (and potential disk IOs) for the requested fields.

To create a clustered index, all that is need is to add "clustering:true" as in the following example (note that version 2 indexes are Fractal Tree Indexes):

```
db.tokubench.ensureIndex({URI : 1}, {v : 2, clustering : true})
```

This benchmark measured the performance of a single threaded insertion workload combined with a range query retrieving 1000 documents greater than or equal to a random URI. The range query runs on a separate thread and sleeps 60 seconds after each completed query.

The inserted documents contained the following: URI (character), name (character), origin (character), creation date (timestamp), and expiration date (timestamp). A total of four secondary indexes were created: URI (clustered), name, origin, and creation date. The benchmark was run with journaling disabled and the default WriteConcern (http://api.mongodb.org/java/current/com/mongodb/WriteConcern.html) disabled.

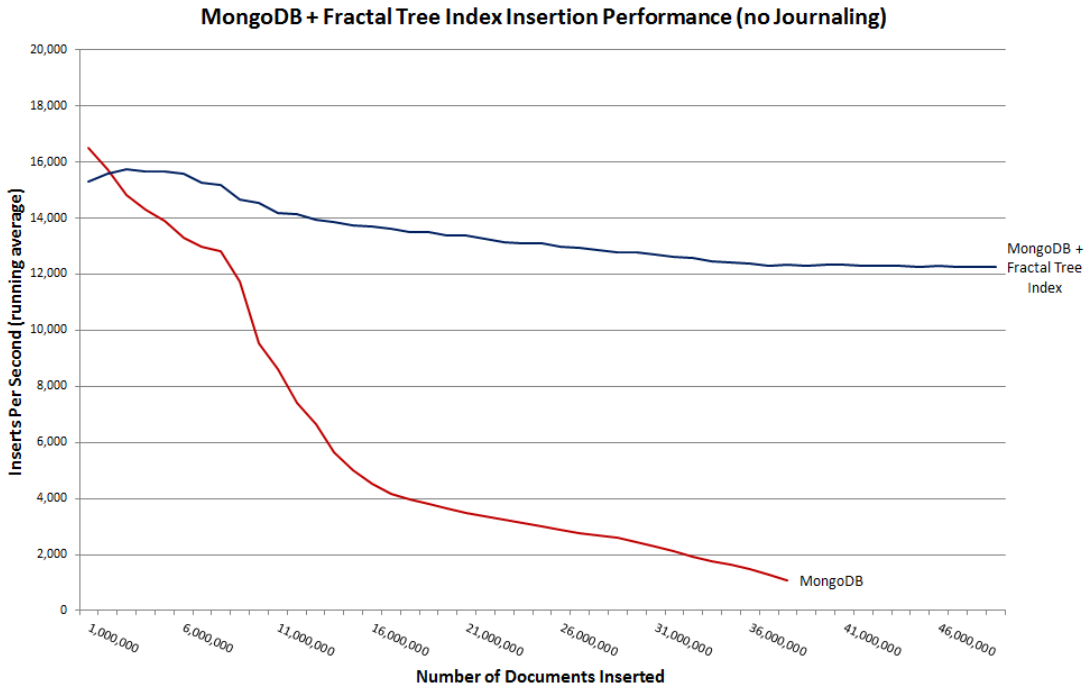
Benchmark Environment

Sun x4150, (2) Xeon 5460, 8GB RAM, StorageTek Controller (256MB, write-back), 4x10K SAS/RAID 0

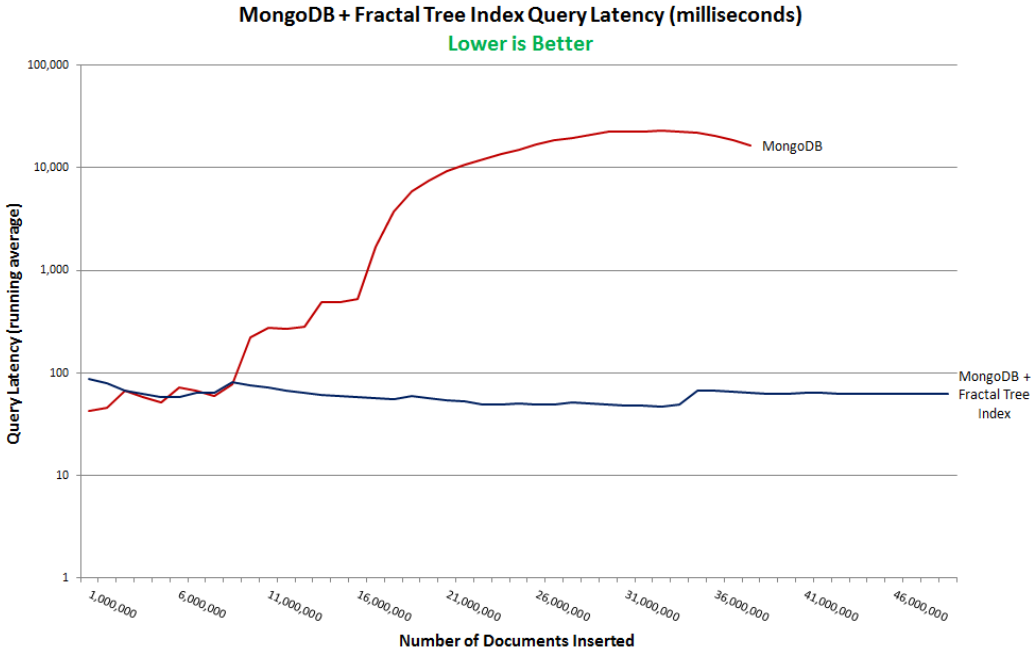
Ubuntu 10.04 Server (64-bit), ext4 file system

MongoDB v2.2.RC0

Benchmark Results



The exit velocity of basic MongoDB is 1,092 inserts per sec at 38 million document insertions versus MongoDB with Fractal Tree Indexes exit velocity of 12,241 inserts per sec at 49 million document insertions: an improvement of **1,020%**.



More interesting is the query performance. Note that this is a latency graph where lower is better and also that the Y-axis is on a log scale to make comparison easier. MongoDB exited with an average of 16,668 milliseconds per query versus MongoDB with Fractal Tree Indexes average of 62 milliseconds: a **26,816%** improvement.

BENCHMARK #3:
Covered Indexes vs. Clustered Fractal Tree Indexes

MongoDB's covered indexes (http://www.mongodb.org/display/DOCS/Retrieving+a+Subset+of+Fields#RetrievingaSubsetofFields-CoveredIndexes) can provide some performance benefits over a regular MongoDB index, as they reduce the amount of IO required to satisfy certain queries. In essence, when all of the fields requested are present in the index key, then MongoDB does not have to go back to the main storage heap to retrieve anything. The benchmark results are further down, but first let's compare MongoDB's Covered Indexes with Tokutek's Clustered Fractal Tree Indexes.

	MongoDB Covered Indexes	**Tokutek Clustered Fractal Tree Indexes**
Query Efficiency	Improved when all requested fields are part of index key	Always improved, all non-keyed fields are stored in the index
Index Size	Data is not compressed	Generally 10x to 20x compression, user selects zlib, quicklz, or lzma. *Note that non-clustered indexes are compressed as well.*
Planning/Maintenance	Index "covers" a fixed set of fields, adding a new field to an existing covered index requires a drop and recreate of the index.	None, all fields in the document are always available in the index.

Covered indexes are really about a well-defined schema, yet NoSQL is often thought of as "schema-less. When a new field covered by an existing index is added to a very large MongoDB collection, the drop and recreate process will take a long time. On the other hand, a clustered Fractal Tree Index will automatically

include this new field so there is no need to drop/recreate unless the field needs to be part of a .find() operation itself.

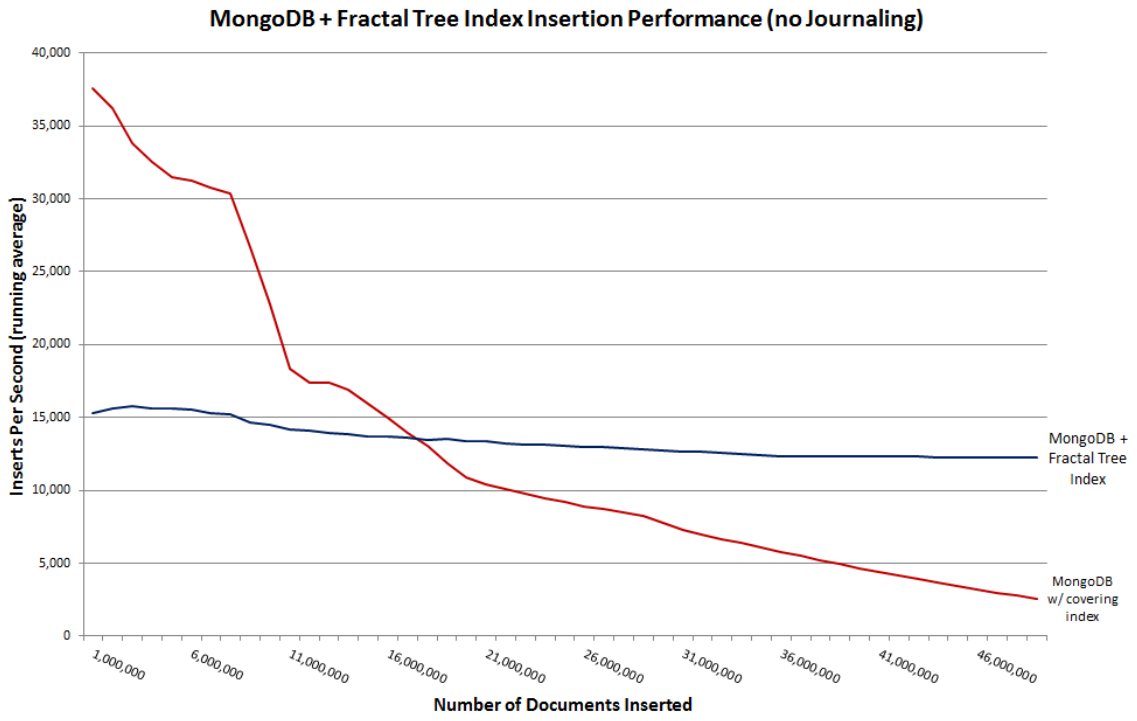
The benchmark previously contained an index on URI, but all queries required a lookup (and IO) to get the rest of the fields. The following change allows lookups by URI to return creation, name, and/or origin without additional IO:

```
// old index, URI only
// db.tokubench.ensureIndex({URI : 1})

// new covered index
db.tokubench.ensureIndex({URI : 1, creation : 1, name : 1, origin : 1})
```

Other than the index change, the benchmark is unchanged. It measured the performance of a single threaded insertion workload combined with a range query retrieving 1000 documents greater than or equal to a random URI. The range query runs on a separate thread and sleeps 60 seconds after each completed query.

The inserted documents contained the following: URI (character), name (character), origin (character), creation date (timestamp), and expiration date (timestamp). A total of four secondary indexes were created: URI (clustered), name, origin, and creation date.



The benchmark ran with journaling disabled and the default WriteConcern disabled.

Benchmark Environment

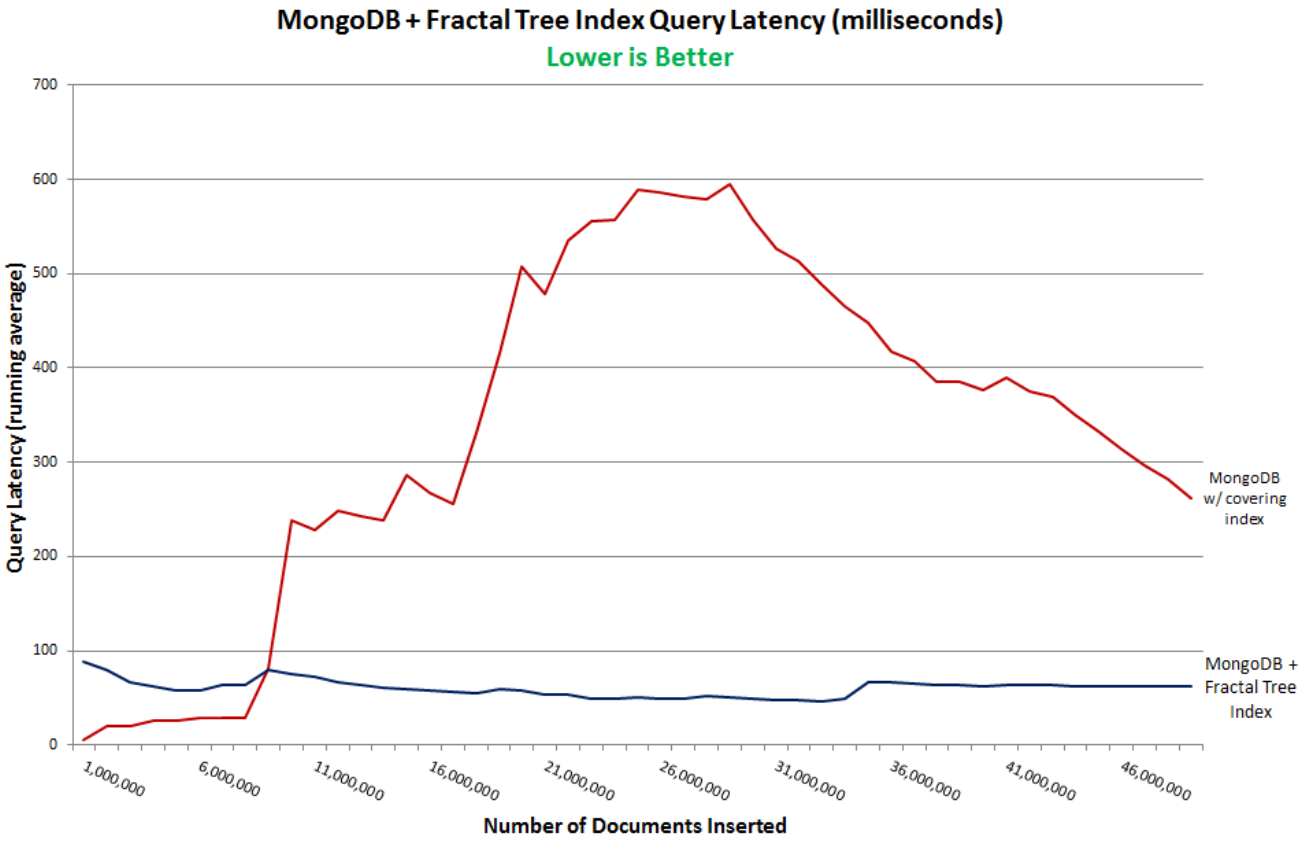
Sun x4150, (2) Xeon 5460, 8GB RAM, StorageTek Controller (256MB, write-back), 4x10K SAS/RAID 0

Ubuntu 10.04 Server (64-bit), ext4 file system

MongoDB v2.2.RC0

Benchmark Results

The exit velocity of standard MongoDB in the preceding graph was 2,594 inserts per second at 49 million document insertions versus MongoDB with Fractal Tree Indexes exit velocity of 12,241 inserts per second at 49 million document insertions: an improvement of **371%**. It is important to note that the insertion performance of standard MongoDB was still in steep decline; most likely it would continue dropping with further insertions.



Note that this is a latency graph where lower is better. MongoDB exited with an average of 262 milliseconds per query versus MongoDB with Fractal Tree Indexes

average of 62 milliseconds: a **322%** improvement. While the shape of the MongoDB graph cannot be explained, it would be interesting to run this experiment out to 100 million inserts.

This benchmark shows that while MongoDB's covered indexes are helpful to the overall performance of the system, the insertion performance is in steady decline and the query performance is inconsistent.

Additional Information

For more information on each of these benchmarks, please visits the corresponding blogs:

- Benchmark #1: 10x Insertion Performance Increase for MongoDB with Fractal Tree Indexes
  - http://www.tokutek.com/2012/08/10x-insertion-performance-increase-for-mongodb-with-fractal-tree-indexes
- Benchmark #2: 268x Query Performance Increase for MongoDB with Fractal Tree Indexes
  - http://www.tokutek.com/2012/08/268x-query-performance-increase-for-mongodb-with-fractal-tree-indexes-say-what
- Benchmark #3: Covered Indexes vs. Clustered Fractal Tree Indexes
  - http://www.tokutek.com/2012/09/mongodb-index-shootout-covered-indexes-vs-clustered-fractal-tree-indexes

Tokutek hopes that sharing these results with the community will elicit people's thoughts on applications where this might help, suggestions for next steps, and any other feedback.

To leave comments or thoughts on our site, please go to (http://www.tokutek.com/2012/09/looking-for-mongodb-users-to-test-fractal-tree-indexing/).

To send us direct inquiries, please contact us at contact@tokutek.com.

About Tokutek

Tokutek's mission is to transform the way data is stored and retrieved and deliver a quantum leap in the performance of databases and file systems.

Technology

Tokutek's patented technology is a result of ten years of research and development by experts in cache-oblivious algorithmics and is used to accelerate key database operations by orders of magnitude. Tokutek's solution is based on a revolutionary new Fractal Tree® technology. The Tokutek solution plugs into the back-end of any database, allowing operating practices to continue without change while the performance of the system improves dramatically.

History

Founded in 2006, Tokutek is a private company located in Lexington, MA. Tokutek was formed to commercialize a new storage technology developed at and licensed from MIT, Stony Brook, and Rutgers.