



TokuDB[®] v5.2.7 with Fractal Tree[®] Indexing
for
MariaDB[®] v5.2.3
User's Guide for linux
January 19, 2012

Notice: Tokutek and TokuDB are registered trademarks and Fractal Tree is a trademark of Tokutek, Inc. MySQL and Sun Microsystems are registered trademarks of Sun Microsystems, Inc.. InnoDB and Oracle are registered trademarks of Oracle Corporation. MariaDB is a registered trademark of Monty Program Ab.

Revision: 39012

1 Introduction

TokuDB, our flagship product, is a highly scalable, zero maintenance downtime, MySQL storage engine that delivers indexing-based query acceleration and enables hot schema modifications. TokuDB is a "drop-in" storage engine requiring no changes to MySQL applications or code and is fully ACID and MVCC compliant.

Providing near seamless compatibility for MariaDB applications, data is loaded, inserted, and queried using standard MySQL commands, with no restrictions or special requirements. Tables can be individually defined to use TokuDB, MyISAM, InnoDB® or other MySQL-compliant storage engines.

Additional features unique to TokuDB include:

- Hot schema changes:
 - Hot index creation: TokuDB tables support insertions, deletions and queries with no down time while indexes are being added to that table
 - Hot column addition, deletion and rename: TokuDB tables support insertions, deletions and queries with minimal down time when an alter table adds, deletes or renames columns .
- Fast recovery time (seconds, not hours or days)
- Immunity to database disorder or "aging" and the gradual performance degradation it causes
- 5x-15x data compression

Please see the Quickstart Guide for instructions on installing and using TokuDB for MariaDB. A list of what's new in this release can be found in Appendix B.

2 Getting the most from TokuDB

Fast insertions and deletions: TokuDB's Fractal Tree technology enables fast indexed insertions and deletions. Fractal Trees match B-trees in their indexing sweet spot — sequential data — and are up to two orders of magnitude faster for random data with high cardinality.

In practice, slow indexing often leads users to choose a smaller number of sub-optimal indexes in order to keep up with incoming data rates. These sub-optimal indexes result in disproportionately slower queries, since the difference in speed between a query with an index and the same query when no index is available can be many orders of magnitude. However, with fast indexing enabled by Fractal Tree technology, we have observed significant in query time, as much as five orders of magnitude. Thus, **fast indexing means fast queries.**

Hot index creation: TokuDB allows the addition of indexes to an existing table while inserts and queries are being performed on that table. This means that MariaDB can be run continuously with no blocking of queries or insertions while indexes are added and eliminates the down-time that index changes would otherwise require.

Hot column addition, deletion and rename: TokuDB allows the addition of new columns to an existing table, the deletion of existing columns from an existing table and the renaming of an existing column while inserts and queries are being performed on that table. MariaDB closes tables during an alter table, but the table becomes available for insertions and queries after seconds to minutes, rather than hours or days of offline alter table.



Compression: TokuDB compresses all data on disk, including indexes. Compression lowers cost by reducing the amount of storage required and frees up disk space for additional indexes to achieve improved query performance. Depending on the compressibility of the data, we have seen compression ratios ranging from 5x to 15x.

No Aging/Fragmentation: TokuDB supports insertions and deletions without exhibiting decreased performance as the database ages. Since TokuDB's Fractal Tree indexes never fragment, the customary practice of dump/reload or `OPTIMIZE TABLE` to restore database performance never needed. This means that MariaDB can be run indefinitely even as the the database is aggressively updated.

Clustering keys and other indexing improvements: TokuDB tables are clustered on the primary key. TokuDB also supports clustering secondary keys, providing better performance on a broader range of queries. A *clustering key* includes (or clusters) all of the columns in a table along with the key. As a result, one can efficiently retrieve any column when doing a range query on a clustering key.

Also, with TokuDB, an auto-increment column can be used in any index and in any position within an index. Lastly, TokuDB indexes can include up to 32 columns.

Bulk loader: TokuDB uses a parallel loader to create tables and offline indexes. This parallel loader will use multiple cores for fast offline table and index creation.

Fast recovery: TokuDB supports very fast recovery, typically less than a minute.

Progress tracking: Running `SHOW PROCESSLIST` when adding indexes provides status on how many rows have been processed. Running `SHOW PROCESSLIST` also shows progress on queries, as well as insertions, deletions and updates. This information is helpful for estimating how long operations will take to complete.

Full-featured Database: TokuDB supports fully ACID-compliant transactions. TokuDB supports MVCC (Multi-Version Concurrency Control) and serialized isolation levels as well as row-level locking. TokuDB scales with high number of client connections, even for large tables.

3 Using TokuDB

3.1 Managing TokuDB Files

It is imperative that no TokuDB file be moved or modified directly by the user. Such modifications will probably result in the need to recover the database from a backup. In order to drop or rename a table or an index, for example, please use the SQL interface.

3.2 Exploit Fast Insertions: Define Richer Indexes

TokuDB's fast indexing enables fast queries through the use of rich indexes, such as covering and clustering indexes. It's worth investing some time to optimize index definitions to get the best performance from MariaDB and TokuDB. Here are some resources to get you started:

- "Understanding Indexing" by Zardosht Kasheff, video and slides:

<http://vimeo.com/26454091>

http://www.tokutek.com/wp-content/uploads/2011/04/understanding_indexing_MySQL_UC.pdf

- <http://www.mysqlperformanceblog.com/2009/06/05/a-rule-of-thumb-for-choosing-column-order-in-indexes/>



- *TokuView Blog*, the Tokutek Blog has many entries on indexing. Please see:

http://tokutek.com/2009/05/covering_indexes_orders_of_magnitude_improvements/

http://tokutek.com/2009/05/introducing_multiple_clustering_indexes/

http://tokutek.com/2009/05/clustering_indexes_vs_covering_indexes/

http://tokutek.com/2009/06/long_index_keys/

http://tokutek.com/2009/06/how_clustering_indexes_sometimes_help_update_and_delete_performance/

as well as newer postings at:

<http://tokutek.com/tokuvview>

- *Covering Indexes in MySQL* by Bradley C. Kuzmaul and Vincenzo Liberatore, MySQL Magazine, Issue 7, Winter 2009. See:

<http://www.paragon-cs.com/mag/issue7.pdf>

as well as some slides of Bradley C. Kuzmaul's talk on covering indexes at the last MySQL users conference. See:

<http://www.tokutek.com/kuzmaul-mysqluc-percona-09-slides.pdf>

- *High Performance MySQL, 2nd Edition*, by Baron Schwartz, Peter Zaitsev, Vadim Tkachenko, Jeremy D. Zawodny, Arjen Lentz and Derek J. Balling, ©2008, O'Reilly Media. See Chapter 3 *Schema Optimization and Indexing*.

One of the keys to exploiting TokuDB's strength in indexing to make use of clustering indexes, described above.

To define a key as clustering, simply add the word `clustering` before the key definition. Some examples:

```
mysql> create table foo (  
-> a int,  
-> b int,  
-> c int,  
-> primary key a_index (a),  
-> clustering key b_index (b))  
-> engine = tokudb;
```

In this example, the primary table is indexed on column `a`. Additionally, there is a secondary clustering index (named `b_index`) sorted on column `b`. Unlike non-clustered indexes, clustering indexes include all the columns of a table and can be used as covering indexes. For example, the query:

```
mysql> select c from foo where b between 10 and 100;
```

will run very fast using the clustering `b_index`. This index is sorted on column `b`, making the `where` clause fast, and includes column `c`, which avoids lookups in the primary table to satisfy the query.

TokuDB makes clustering indexes feasible because of its excellent compression and very high indexing rates. For more details on using clustering indexes, see:

http://tokutek.com/2009/05/introducing_multiple_clustering_indexes/



3.3 Hot index creation

TokuDB allows you to add indexes to an existing table and still perform inserts and queries on that table while the index is being created.

The `ONLINE` keyword is not used. Instead, the value of the `tokudb_create_index_online` client session variable is examined. See Client Session Variables section 4.1.7.

Hot index creation is invoked using the `CREATE INDEX` command after setting `tokudb_create_index_online=on`.

For example:

```
mysql> set tokudb_create_index_online=on;
Query OK, 0 rows affected (0.00 sec)
```

```
mysql> create index example_idx on example_tbl (example_field);
```

Alternatively, using the `ALTER TABLE` command for creating an index will create the index offline (with the table unavailable for inserts or queries), regardless of the value of `tokudb_create_index_online`. The only way to hot create an index is to use the `CREATE INDEX` command.

Hot creating an index will be slower than creating the index offline, and progress depends how busy the `mysqld` server is with other tasks. Progress of the index creation can be seen by using the `SHOW PROCESSLIST` command (in another client). Once the index creation completes, the new index will be used in future query plans.

If more than one hot `CREATE INDEX` is issued for a particular table, the indexes will be created serially. An index creation that is waiting for another to complete will be shown as `Locked` in `SHOW PROCESSLIST`. We recommend that each `CREATE INDEX` be allowed to complete before the next one is started.

3.4 Hot Column Addition, Deletion and Renaming (HCADR)

TokuDB allows you to add or delete columns to an existing table or rename an existing column in a table with little blocking of other updates and queries. Typically HCADR blocks other queries with a table lock for only a few seconds or minutes. After that initial short-term table locking, the system modifies each row (when adding or deleting columns) later, when the row is next brought into main memory from disk. For column rename, all the work is done during the seconds of downtime. On-disk rows need not be modified.

To get good performance from HCADR, observe the following guidelines.

- The work of altering the table for column addition or deletion is performed as subsequent queries and insertions touch the parts of the Fractal Tree, both in the primary index and in each clustering index. In some cases, this work may temporarily cause queries or insertions to slow down.

You can force the column addition or deletion work to be performed all at once, by performing an `OPTIMIZE TABLE`, using the standard syntax of `OPTIMIZE TABLE X`, when a column has been added to or deleted from table `X`. It is important to note that in TokuDB v5.2.7, `OPTIMIZE TABLE` is also hot, so that a table supports updates and queries without blocking while an `OPTIMIZE TABLE` is being performed. Also, a hot `OPTIMIZE TABLE` does not rebuild the indexes, since TokuDB indexes do not age. Rather, they flush all background work, such as that induced by a hot column addition or deletion.

- Avoid mixing column addition and deletion in one statement. You can quickly add as many columns as you want in one statement, or you can quickly delete as many columns as you want in one statement, but you cannot do both quickly in one statement. If you want to add and delete columns, use two statements, one to add, and one to delete columns.
- Avoid adding or deleting a column at the same time as adding or dropping an index.





- The time that the table lock is held can vary. The table-locking time for HCADR is dominated by the time it takes to flush dirty pages, because MariaDB closes the table after altering it. If a checkpoint has happened recently, this operation is fast – on the order of seconds. However, if the table has many dirty pages, then the flushing stage can take on the order of minutes.
- Avoid dropping a column that is part of an index. If a column to be dropped is part of an index, then dropping that column is slow. To drop a column that is part of an index, first drop the indexes that reference the column in one alter table statement, and then drop the column in another statement.
- Only rename one column per statement. Renaming more than one column will revert to the standard MariaDB blocking behavior. The proper syntax is `ALTER TABLE <table-name> CHANGE <old-column-name> <new-column-name> <data-type> <required-ness> <default>`, for example, `ALTER TABLE foo CHANGE col1 column1 INT(10) NOT NULL;`. Note that all column attributes must be specified. `ALTER TABLE foo CHANGE col1 column1;` induces a slow, blocking column rename.
- Hot column rename does not support the following data types: `TIME`, `ENUM`, `BLOB`, `TINYBLOB`, `MEDIUMBLOB`, `LOB`. Renaming columns of these types will revert to the standard MariaDB blocking behavior.
- Temporary tables cannot take advantage of HCADR. Temporary tables are typically small anyway, so altering them using the standard method is usually fast.

3.5 Transactions, and ACID-compliant Recovery

By default, TokuDB checkpoints all open tables regularly and logs all changes between checkpoints, so that after a power failure or system crash, TokuDB will restore all tables into their fully ACID-compliant state. That is, all committed transactions will be reflected in the tables, and any transaction not committed at the time of failure will be rolled back.

The default checkpoint period is 60 seconds, and this specifies the time from the end of one checkpoint to the beginning of the next. It is also related to the frequency with which log files are trimmed, as described below. The user can induce a checkpoint at any time by issuing a `flush logs` command. When a database is shut down normally it is also checkpointed and all open transactions are aborted. The logs are trimmed at startup.

3.5.1 Managing Log Size

TokuDB keeps log files back to the most recent checkpoint. Whenever a log file reaches 100MB, a new log file is started. Whenever there is a checkpoint, all log files older than the checkpoint are discarded. If the checkpoint period is set to be a very large number, logs will get trimmed less frequently. This value is set to 60s by default.

TokuDB also keeps rollback logs for each open transaction. The size of each log is proportional to the amount of work done by its transaction and is stored compressed on disk. Rollback logs are trimmed when the associated transaction completes.

3.5.2 Recovery

Recovery is fully automatic with TokuDB. TokuDB uses both the log files and rollback logs to recover from a crash. The time to recover from a crash is proportional to the combined size of the log files and uncompressed size of rollback logs. Thus, if there were no long-standing transactions open at the time of the the most recent checkpoint, recovery will take less than a minute.

3.5.3 Disabling the Write Cache

When using any transaction-safe database, it is essential that you understand the write-caching characteristics of your hardware. TokuDB provides transaction safe (ACID compliant) data storage for MariaDB. However, if the underlying operating system or hardware does not actually write data to disk when it says it did, the system can corrupt your database when the machine crashes. For example, TokuDB can not guarantee proper recovery if it is mounted on an NFS volume. It is always safe to disable the write cache, but you may be giving up some performance.

For most configurations you must disable the write cache on your disk drives. On ATA/SATA drives a command such as

```
hdparm -W0 /dev/hda
```

may disable the write cache.

There are some cases where you can keep the write cache, for example:

- Write caching can remain enabled when using XFS, but only if XFS reports that disk write barriers work. If you see one of the following messages in `/var/log/messages`, then you must disable the write cache:
 - “Disabling barriers, not supported with external log device”
 - “Disabling barriers, not supported by the underlying device”
 - “Disabling barriers, trial barrier write failed”

XFS write barriers appear to succeed for single disks (with no LVM), or for very recent kernels (such as that provided by Fedora 12). For more details, see http://xfs.org/index.php/XFS_FAQ#Q:_How_can_I_tell_if_I_have_the_disk_write_cache_enabled.3F

In some cases, you must disable the write cache:

- If you use the `ext3` file system, you must disable the write cache.
- If you use LVM, you must disable the write cache. (Although recent Linux kernels, such as Fedora 12, have fixed this problem.)
- If you use Linux’s software RAID, you must disable the write cache.
- If you use a RAID controller with battery-backed-up memory, you must disable the write cache. This may seem counterintuitive. See the `xfs.org` link above for details.

In summary, you should disable the write cache unless you have a very specific reason not to do so.

3.6 Progress Tracking

TokuDB has a system for tracking progress of long running statements, thereby removing the need to define triggers to track statement execution, as follows:

Bulk Load: When loading large tables using `LOAD DATA INFILE` commands, doing a `SHOW PROCESSLIST` command in a separate client session shows progress. There are two progress stages. The first will state something like “Inserted about 1000000 rows”. After all rows are processed like this, the next stage tracks progress by showing what fraction of the work is done (e.g. “Loading of data about 45% done”)



Adding Indexes: When adding indexes via `ALTER TABLE` or `CREATE INDEX`, the command `SHOW PROCESSLIST` shows progress. When adding indexes via `ALTER TABLE` or `CREATE INDEX`, the command `SHOW PROCESSLIST` will include an estimate of the number of rows processed use this information to verify progress is being made. Similar to bulk loading, the first stage shows how many rows have been processed, and the second stage shows progress with a fraction.

Commits and Aborts: When committing or aborting a transaction, the command `SHOW PROCESSLIST` will include an estimate of the transactional operations processed.

3.7 Migrating to TokuDB

To convert an existing table to use the TokuDB engine, run "alter table ... engine=TokuDB". If you wish to load from a file, use "load data infile" and not `mysqldump`. Using `mysqldump` will be much slower. To create a file that can be loaded with "load data infile", refer to the 'into outfile' option of the syntax for select, <http://dev.mysql.com/doc/refman/5.1/en/select.html>. Note that creating this file does NOT save the schema of your table, so you may want to create a copy of that as well.



4 TokuDB Variables

Like all storage engines, TokuDB has variables to tune performance and control behavior. *Fractal Tree algorithms are designed for near optimal performance and TokuDB's default settings should work well in most situations, eliminating the need for complex and time consuming tuning in most cases.*

4.1 Client Session Variables

4.1.1 `unique_checks`

For tables with unique keys, every insertion into the table causes a lookup by key followed by an insertion, if the key is not in the table. This greatly limits insertion performance. If one knows by design that the rows being inserted into the table have unique keys, then one can disable the key lookup prior to insertion as follows:

```
set unique_checks=off;
```

If your primary key is an auto-increment key, and none of your secondary keys are declared to be unique, then setting `unique_checks` to `off` will provide limited performance gains. On the other hand, if your primary key has a lot of entropy (it looks random), or your secondary keys are declared unique and have a lot of entropy, then disabling unique checks can provide a significant performance boost.

If `unique_checks` is set to `off` when the primary key is not unique, secondary indexes may become corrupted. In this case, the indexes should be dropped and rebuilt. This behavior differs from that of InnoDB, in which uniqueness is always checked on the primary key, and setting `unique_checks` to `off` turns off uniqueness checking on secondary indexes only. Turning off uniqueness checking on the primary key can provide large performance boosts, but it should only be done when the primary key is known to be unique.

4.1.2 `tokudb_commit_sync`

Session variable `tokudb_commit_sync` controls whether or not the transaction log is flushed when a transaction commits. The default behavior is that the transaction log is flushed by the commit. Flushing the transaction log requires a disk write and may adversely affect the performance of your application.

To disable synchronous flushing of the transaction log, set the `tokudb_commit_sync` session variable as follows:

```
set tokudb_commit_sync=off;
```

Setting this parameter to `off` may make the system run faster. However, transactions committed since the last checkpoint are not guaranteed to survive a crash.

4.1.3 `tokudb_lock_timeout`

Session variable `tokudb_lock_timeout` controls the amount of time that a transaction will wait for a lock held by another transaction to be released. If the conflicting transaction does not release the lock within the lock timeout, the transaction that was waiting for the lock will get a lock timeout error. The units are milliseconds. A value of 0 disables lock waits. The default value is 4000 (four seconds).

If your application gets a "lock wait timeout" error (-30994), then you may find that increasing the `tokudb_lock_timeout` may help. If your application get a "deadlock found" error (-30995), then you need to abort the current transaction and retry it.

This variable replaces `tokudb_read_lock_wait` and `tokudb_write_lock_wait`, which are now deprecated.

4.1.4 `tokudb_pk_insert_mode`

This session variable controls the behavior of primary key insertions with the command `REPLACE INTO` and `INSERT IGNORE` on tables with no secondary indexes and on tables whose secondary keys whose every column is also a column of the primary key. For instance, the table `(a int, b int, c int, primary key (a,b), key (b))` is affected, because the only column in the key of 'b' is present in the primary key. TokuDDB can make these insertions really fast on these tables. However, triggers may not work and row based replication definitely will not work in this mode. This variable takes the following values, to control this behavior. This only applies to tables described above, using the command `REPLACE INTO` or `INSERT IGNORE`. All other scenarios are unaffected.

- 0: Insertions are fast, regardless of whether triggers are defined on the table. `REPLACE INTO` and `INSERT IGNORE` statements fail if row based replication is enabled.
- 1: (default) - insertions are fast, if there are no triggers defined on the table. Insertions may be slow if triggers are defined on the table. `REPLACE INTO` and `INSERT IGNORE` statements fail if row based replication is enabled.
- 2: insertions are slow, all triggers on the table work, and row based replication works on `REPLACE INTO` and `INSERT IGNORE` statements.

4.1.5 `tokudb_load_save_space`

This session variable changes the behavior of the bulk loader. When it is `off`, bulk loads are faster but use more temporary disk space. When it is `on`, the opposite is the case: bulk loads are replaced by iterative inserts, which are still fast, but not as fast. This saves disk space. The default value is `off`. Note, the location of the temporary disk space used by the bulk loader may be specified with the `tokudb_tmp_dir` server variable.

If a `load data infile` statement fails with the error message `ERROR 1030 (HY000): Got error 1 from storage engine` then there may not be enough disk space for the optimized loader, so turn on `tokudb_load_save_space` and try again.



See Appendix A section "Uninformative error message."

4.1.6 `tokudb_prelock_empty`

By default, in 5.2.7, TokuDB preemptively grabs an entire table lock on empty tables. If one transaction is doing the loading, such as when the user is doing a table load into an empty table, this default provides a considerable speedup. However, if multiple transactions try to do concurrent operations on an empty table, all but one transaction will be locked out. Setting `tokudb_prelock_empty = off` optimizes for this multi-transaction case by turning off preemptive prelocking. Note that if this variable is set to `off`, fast bulk loading is turned off as well.

4.1.7 `tokudb_create_index_online`

This variable controls whether indexes created with the `CREATE INDEX` command are hot (if set `on`), or offline (if set `off`). Hot index creation means that the table is available for inserts and queries while the index is being created. Offline index creation means that the table is not available for inserts and queries while the index is being created.

The default value for `tokudb_create_index_online` is `off`.

4.1.8 `tokudb_disable_slow_alter`

This variable controls whether slow alter tables are allowed. For example, `ALTER TABLE foo ADD COLUMN a int, DROP COLUMN b;` is slow because HCADR does not allow a mixture of column addition and deletions. Note, however, that multiple columns may be added in a single statement with minimal performance impact. Similarly, multiple columns may be deleted in a single statement. In order to achieve fast performance with minimal down time, for the example above we recommend `ALTER TABLE foo ADD COLUMN a int; SELECT COUNT (*) FROM foo FORCE INDEX(primary); ALTER TABLE foo DROP COLUMN b;` Setting `tokudb_disable_slow_alter` to `on` will block the former command. This prevents a user from accidentally performing a slow alter table that would lock the table for a long time.

The default value `tokudb_disable_slow_alter` is `off`.

4.1.9 `tokudb_read_block_size`

Fractal tree leaves are subdivided into read blocks, in order to speed up point queries. The session variable `tokudb_read_block_size` controls the target uncompressed size of the read blocks. The units are bytes. The default is 131,072 (128KB). The minimum value of this variable is 4096.

Changing the value of `tokudb_read_block_size` only affects subsequently created tables. The value of this variable cannot be changed in an existing table without a dump and reload.

4.1.10 `tokudb_read_buf_size`

This variable controls the size of the buffer used to store values that are bulk fetched as part of a large range query. Its unit is bytes and its default value is 131,072 (128KB). A value of 0 turns off bulk fetching. Each client keeps a thread of this size, so it should be lowered if situations where there are a large number of clients simultaneously querying a table.



4.1.11 `tokudb_disable_prefetching`

TokuDB attempts to aggressively prefetch additional blocks of rows, which is helpful for most range queries but may create unnecessary IO for range queries with LIMIT clauses. Prefetching is on by default, with a value of 0, and can be disabled by setting this variable to 1.

4.2 MariaDB Server Variables

4.2.1 `tokudb_cache_size`

This variable configures the size in bytes of the TokuDB cache table. For 64 bit Linux, the default cache table size is 1/2 of physical memory. **Tokutek highly recommends using the default setting in most situations.** Unlike with other storage engines, changing this parameter for TokuDB is unlikely to improve performance.

- Consider decreasing `tokudb_cache_size` if excessive swapping is causing performance problems. Swapping may occur when running multiple mysql server instances or if other running applications use large amounts of physical memory.

4.2.2 `tokudb_data_dir`

This variable configures the directory name where the TokuDB tables are stored. The default location is the MariaDB data directory.

4.2.3 `tokudb_log_dir`

This variable configures the directory name where the TokuDB log files are stored. The default location is the MariaDB data directory. Configuring a separate `log_dir` is somewhat involved. Please contact Tokutek support for more details.

4.2.4 `tokudb_tmp_dir`

This variable configures the directory name where the TokuDB bulk loader stores temporary files. The bulk loader can create large temporary files while it is loading a table, so putting these temporary files on a disk separate from the data directory can be useful. For example, it can make sense to use a high-performance disk for the data directory and a very inexpensive disk for the temporary directory. The default location for temporary files is the MariaDB data directory.

4.2.5 `tokudb_checkpointing_period`

This variable specifies the time in seconds between the end of one checkpoint and the beginning of another. The default time between TokuDB checkpoints is 60 seconds. We recommend leaving this variable unchanged.

4.2.6 `tokudb_write_status_frequency` and `tokudb_read_status_frequency`

TokuDB shows statement progress of queries, inserts, deletes, and updates in `show processlist`. Queries are defined as reads, and inserts, deletes, and updates are defined as writes. Progress for updates is controlled by `tokudb_write_status_frequency`, which is set to 1000, that is, progress is measured every 1000 writes. Progress for reads is controlled by `tokudb_read_status_frequency` which is set to



10,000. For slow queries, it can be helpful to set these variables to 1, and then run `show processlist` several times to understand what progress is being made.

4.2.7 `tokudb_fs_reserve_percent`

This variable controls the percentage of the file system that must be available for inserts to be allowed. By default, this is set to 5. We recommend that this reserve be at least half the size of your physical memory. See FAQ section 5.3 for more information.

4.2.8 `tokudb_cleaner_period`

This variable specifies how often in seconds the cleaner thread runs. The default value is 1. Setting this variable to 0 turns off cleaner threads.

4.2.9 `tokudb_cleaner_iterations`

This variable specifies how many internal nodes get processed in each `tokudb_cleaner_period` period. The default value is 5. Setting this variable to 0 turns off cleaner threads.



5 FAQ

5.1 User Data

Q: How do I determine how much user data TokuDB is storing so I can calculate pricing?

A: TokuDB for MariaDB pricing is based on uncompressed user data exclusive of secondary indexes (please note that secondary indexes are free and that your compressed data on disk will usually be much smaller than your uncompressed data). Non-master replicas are discounted 50% and BLOB data is heavily discounted. See <http://www.tokutek.com/products/pricing/> for TokuDB pricing details.

The amount of uncompressed user data, exclusive of secondary indexes, is available as an estimate or as an exact number via the information schema as tables `TokuDB_user_data` and `TokuDB_user_data_exact` respectively.

The commands for getting the estimate of user data and the exact amount of user data are:

```
select * from information_schema.TokuDB_user_data;
```

and

```
select * from information_schema.TokuDB_user_data_exact;
```

Please note that running the exact command can take a long time.

5.2 Transactional Operations

Q: What transactional operations does TokuDB support?

A: TokuDB supports `BEGIN TRANSACTION`, `END TRANSACTION`, `COMMIT`, `ROLLBACK`, `SAVEPOINT`, and `RELEASE SAVEPOINT`.

5.3 Full Disks

Q: What happens when the disk system fills up?

A: The disk system may fill up during bulk load operations, such as `load data infile` or `create index`, or during incremental operations like `insert`.

In the bulk case, running out of disk space will cause the statement to fail with the error message `ERROR 1030 (HY000): Got error 1 from storage engine`. The temporary space used by the bulk loader will be released. If this happens, you can use a separate physical disk for the temporary files (see section on server variable `tokudb_tmp_dir`) or instruct the bulk loader to not create temporary files by setting the session variable `tokudb_load_save_space` to `on`.

Otherwise, disk space can run low during non-bulk operations. When available space is below a user-configurable reserve (5% by default) inserts are prevented and transactions that perform inserts are aborted. If the disk becomes completely full then TokuDB will freeze until some disk space is made available.



Details

1. There is a free-space reserve requirement, which is a user-configurable parameter given as a percentage of the total space in the file system. The default reserve is five percent. This value is available in the global variable `tokudb_fs_reserve_percent`. We recommend that this reserve be at least half the size of your physical memory.

TokuDB polls the file system every five seconds to determine how much free space is available. If the free space dips below the reserve, then further table inserts are prohibited. Any transaction that attempts to insert rows will be aborted. Inserts are re-enabled when twice the reserve is available in the file system (so freeing a small amount of disk storage will not be sufficient to resume inserts). Warning messages are sent to the system error log when free space dips below twice the reserve and again when free space dips below the reserve.

Even with inserts prohibited it is still possible for the file system to become completely full. For example this can happen because another storage engine or another application consumes disk space.

2. If the file system becomes completely full, then TokuDB will freeze. It will not crash, but it will not respond to most SQL commands until some disk space is made available. When TokuDB is frozen in this state, it will still respond to the SQL command `SHOW ENGINE TOKUDB STATUS`. Making disk space available will allow the storage engine to continue running, but inserts will still be prohibited until twice the reserve is free.

Note, engine status now displays a field indicating if disk free space is above twice the reserve, below twice the reserve, or below the reserve. It will also display a special warning if the disk is completely full.

3. In order to make space available on this system you can:

- Add some disk space to the file system.
- Delete some non-TokuDB files manually.
- If the disk is not completely full, you may be able to reclaim space by aborting any transactions that are very old. Old transactions can consume large volumes of disk space in the recovery log.
- If the disk is not completely full, you can drop indexes or drop tables from your TokuDB databases.

- Deleting large numbers of rows from an existing table and then closing the table may free some space, but it may not. Deleting rows may simply leave unused space (available for new inserts) inside TokuDB data files rather than shrink the files (internal fragmentation).

The fine print

1. The TokuDB storage engine can use up to three separate file systems simultaneously, one each for the data, the recovery log, and the error log. All three are monitored, and if any one of the three falls below the relevant threshold then a warning message will be issued and inserts may be prohibited.
2. Warning messages to the error log are not repeated unless available disk space has been above the relevant threshold for at least one minute. This prevents excess messages in the error log if the disk free space is fluctuating around the limit.
3. Even if there are no other storage engines or other applications running, it is still possible for TokuDB to consume more disk space when operations such as row delete and query are performed, or when checkpoints are taken. This can happen because TokuDB can write cached information when it is time-efficient rather than when inserts are issued by the application, because operations in addition to insert (such as delete) create log entries, and also because of internal fragmentation of TokuDB data files.
4. The variable `tokudb_fs_reserve_percent` can not be changed once the system has started. It can only be set in `my.cnf` or on the `mysqld` command line.



5.4 Backup

Q: How do I back up a system with TokuDB tables?

A: TokuDB tables are represented in the file system with dictionary files, log files, and metadata files. A consistent copy of all of these files must be made during a backup. Copying the files while they may be modified by a running MariaDB may result in an inconsistent copy of the database.

LVM snapshots may be used to get a consistent snapshot of all of the TokuDB files. The LVM snapshot may then be backed up at leisure.

The `select into outfile` statement may be used to get a logical backup of the database. Since the TokuDB has a parallel loader, the time to load the logical snapshot may be several times faster than previous TokuDB releases.

References

The MySQL 5.1 reference manual describes several backup methods and strategies. In addition, we recommend reading the backup and recovery chapter in the following book.

High Performance MySQL, 2nd Edition, by Baron Schwartz, Peter Zaitsev, Vadim Tkachenko, Jeremy D. Zawodny, Arjen Lentz and Derek J. Balling, ©2008, O'Reilly Media.

Cold Backup

When MariaDB is shutdown, a copy of the MariaDB data directory, the TokuDB data directory, and the TokuDB log directory can be made. In the simplest configuration, the TokuDB files are stored in the MariaDB data directory with all of other MariaDB files. One merely has to back up this directory.

Hot Backup using mylvmbackup

The `mylvmbackup` program, located on <https://launchpad.net/>, works with TokuDB. We have been using version 0.13 at Tokutek. It does all of the magic required to get consistent copies of all of the MariaDB tables, including MyISAM tables, InnoDB tables, etc., creates the LVM snapshots, and backs up the snapshots.

Logical snapshots

A logical snapshot of the databases uses a SQL statements to retrieve table rows and restore them. When used within a transaction, a consistent snapshot of the database can be taken. This method can be used to export tables from one database server and import them into another server.

The `select into outfile` statement is used to take a logical snapshot of a database. The `load data infile` statement is used to load the table data. Please see the MySQL 5.1 reference manual for details.

mysqlhotcopy

Please do not use `mysqlhotcopy` to back up TokuDB tables. This script is incompatible with TokuDB.



5.5 Show Engine Status

Q: What information is provided by `show engine tokudb status`?

A: Engine status is primarily intended to provide Tokutek support with information about the internal operations of the database. Please include the output of `show engine tokudb status` whenever sending mail to Tokutek support.

Engine status also provides information we think may be useful to you. The fields of engine status are described here:

`Version`: TokudB version string.

`disk free space`: This is a gross estimate of how much of your file system is available. Possible displays in this field are:

- more than twice the reserve (for example, “more than 10 percent of total file system space”)
- less than twice the reserve
- less than the reserve
- file system is completely full

`time of environment creation`: This is the time when the TokudB storage engine was first started up. Normally, this is when `mysqld` was initially installed with TokudB 5.x. If the environment was upgraded from TokudB 4.x (4.2.0 or later), then this will be displayed as “Dec 31, 1969” on Linux hosts.

`time of engine startup`: This is the time when the TokudB storage engine started up. Normally, this is when `mysqld` started.

`time now`: This is the current time on the host machine where `mysqld` is running.

`checkpoint period`: This is the interval in seconds between the end of an automatic checkpoint and the beginning of the next automatic checkpoint.

`checkpoint status code (0 = idle)`: This is a code number indicating the current state of the checkpoint process. If this number is non-zero, then a checkpoint is in progress.

`last checkpoint began`: This is the time the last checkpoint began. If a checkpoint is currently in progress, then this time may be later than the time the last checkpoint completed. (Note, if no checkpoint has ever taken place, then this value will be “Dec 31, 1969” on Linux hosts.)

`last complete checkpoint began`: This is the time the last complete checkpoint started. Any data that changed after this time will not be captured in the checkpoint.

`last complete checkpoint ended`: This is the time the last complete checkpoint ended.

`last complete checkpoint LSN`: This is the Log Sequence Number of the last complete checkpoint.

`checkpoints taken`: This is the number of complete checkpoints that have been taken.

`checkpoints failed`: This is the number of checkpoints that have failed for any reason.

`cleaner period`: TokudB includes a cleaner thread that optimizes indexes in the background. This variable is the time, in seconds, between the completion of a group of cleaner operations and the beginning of the next group of cleaner operations. The cleaner operations run on a background thread performing work that does not need to be done on the client thread.

`cleaner iterations`: This is the number of cleaner operations that are performed every cleaner period.



`txn begin`: This is the number of transactions that have been started.

`txn commits`: This is the total number of transactions that have been committed.

`txn aborts`: This is the total number of transactions that have been aborted.

`txn close (commit+abort)`: This is the number of transaction that have been closed. This will normally be the sum of the number of transactions committed plus the number of transactions aborted. (The number may be off slightly because of the way the information is collected.)

`txn_num_open`: This is the number of transactions currently in progress. (The number may be off slightly because of the way the information is collected.)

`txn_max_open`: This is the maximum number of transactions that have been in progress simultaneously.

`txn oldest live`: This is the transaction id of the oldest living transaction, the oldest transaction that has not yet been committed or aborted.

`txn oldest starttime`: This is the time the oldest living transaction began.

`next LSN`: This is the next unassigned Log Sequence Number. It will be assigned to the next entry in the recovery log.

`dictionary inserts`: This is the total number of rows that have been inserted into all primary and secondary indexes combined, when those inserts have been done with a separate recovery log entry per index. For example, inserting a row into a table with one primary and two secondary indexes will increase this count by three, if the inserts were done with separate recovery log entries.

`dictionary inserts fail`: This is the number of single-index insert operations that failed.

`dictionary deletes`: This is the total number of rows that have been deleted from all primary and secondary indexes combined, if those deletes have been done with a separate recovery log entry per index.

`dictionary deletes fail`: This is the number of single-index delete operations that failed.

`dictionary updates`: This is the total number of rows that have been updated in all primary and secondary indexes combined, if those updates have been done with a separate recovery log entry per index.

`dictionary updates fail`: This is the number of single-index update operations that failed.

`dictionary broadcast updates`: This is the number of broadcast updates that have been successfully performed. A broadcast update is an update that affects all rows in a dictionary.

`dictionary broadcast updates fail`: This is the number of broadcast updates that have failed.

`leafentry updates`: This is the number of rows that have been individually updated.

`leafentry broadcast updates`: This is the number of rows that have been updated when the entire dictionary was updated (for example, when the schema has been changed).

`descriptor_set`: This is the number of time a descriptor was updated when the entire dictionary was updated (for example, when the schema has been changed).

`partial_fetch_hit`: The number of times a partial node was present in cache.

`partial_fetch_miss`: The number of times a node was present in cache, but the requested partition was absent.

`partial_fetch_compressed`: The number of times a partial node was present but compressed.



`partial_evictions_nonleaf`: The number of times a partition of a nonleaf node was evicted from cache.

`partial_evictions_leaf`: The number of times a partition of a nonleaf node was evicted from cache.

`msn_discards`: The number of messages that were ignored by a leaf because it had already been applied.

`max_workdone`: The maximum workdone value of any message buffer.

`total_searches`: The total number of search operations performed.

`total_retries`: Total number of search retries due to `TRY_AGAIN`.

`max_search_excess_retries`: Max number of excess search retries (retries - treeheight) due to `TRY_AGAIN`.

`max_search_root_retries`: Max number of times root node was fetched in a single search.

`search_root_retries`: Number of searches that required the root node to be fetched more than once.

`search_tries_gt_height`: Number of searches that required more tries than the height of the tree.

`search_tries_gt_heightplus3`: Number of searches that required more tries than the height of the tree plus three.

`cleaner_iterations`: Total number of times the cleaner thread loop has executed.

`cleaner_total_nodes`: Total number of nodes whose buffers are potentially flushed by cleaner thread.

`cleaner_h1_nodes`: Number of nodes of height one whose message buffers are flushed by cleaner thread.

`cleaner_hgt1_nodes`: Number of nodes of height > 1 whose message buffers are flushed by cleaner thread.

`cleaner_empty_nodes`: Number of nodes that are selected by cleaner, but whose buffers are empty.

`cleaner_nodes_dirtied`: Number of nodes that are made dirty by the cleaner thread.

`cleaner_max_buffer_size`: Max number of bytes in message buffer flushed by cleaner thread.

`cleaner_min_buffer_size`: Min number of bytes in message buffer flushed by cleaner thread.

`cleaner_total_buffer_size`: Total number of bytes in message buffers flushed by cleaner thread.

`cleaner_max_buffer_workdone`: Max workdone value of any message buffer flushed by cleaner thread.

`cleaner_min_buffer_workdone`: Min workdone value of any message buffer flushed by cleaner thread.

`cleaner_total_buffer_workdone`: Total workdone value of message buffers flushed by cleaner thread.

`flush_total`: Total number of flushes done by flusher threads or cleaner threads.

`flush_in_memory`: Number of in memory flushes.

`flush_needed_io`: Number of flushes that had to read a child (or part) off disk.



`flush_cascades`: Number of flushes that triggered another flush in the child.

`flush_cascades_1`: Number of flushes that triggered 1 cascading flush.

`flush_cascades_2`: Number of flushes that triggered 2 cascading flushes.

`flush_cascades_3`: Number of flushes that triggered 3 cascading flushes.

`flush_cascades_4`: Number of flushes that triggered 4 cascading flushes.

`flush_cascades_5`: Number of flushes that triggered 5 cascading flushes.

`flush_cascades_gt_5`: Number of flushes that triggered more than 5 cascading flushes.

`disk_flush_leaf`: Number of leaf nodes flushed to disk, not for checkpoint.

`disk_flush_nonleaf`: Number of nonleaf nodes flushed to disk, not for checkpoint.

`disk_flush_leaf_for_checkpoint`: Number of leaf nodes flushed to disk for checkpoint.

`disk_flush_nonleaf_for_checkpoint`: Number of nonleaf nodes flushed to disk for checkpoint.

`create_leaf`: Number of leaf nodes created.

`create_nonleaf`: Number of nonleaf nodes created.

`destroy_leaf`: Number of leaf nodes destroyed.

`destroy_nonleaf`: Number of nonleaf nodes destroyed.

`split_leaf`: Number of leaf nodes split.

`split_nonleaf`: Number of nonleaf nodes split.

`merge_leaf`: Number of times leaf nodes are merged.

`merge_nonleaf`: Number of times nonleaf nodes are merged.

`dirty_leaf`: Number of times leaf nodes are dirtied when previously clean.

`dirty_nonleaf`: Number of times nonleaf nodes are dirtied when previously clean.

`balance_leaf`: Number of times a leaf node is balanced.

`msg_bytes_in`: How many bytes of messages injected at root (for all trees).

`msg_bytes_out`: How many bytes of messages flushed from h1 nodes to leaves.

`msg_bytes_curr`: How many bytes of messages currently in trees (estimate).

`msg_bytes_max`: How many bytes of messages currently in trees (estimate).

`msg_num`: How many messages injected at root.

`msg_num_broadcast`: How many broadcast messages injected at root.

`num_basements_decompressed_aggressive`: Number of basement nodes decompressed by queries aggressively.

`num_basements_decompressed_normal`: Number of basement nodes decompressed for queries.

`num_basements_decompressed_prefetch`: Number of basement nodes decompressed by a prefetch thread.

`num_basements_decompressed_write`: Number of basement nodes decompressed for writes.

`num_msg_buffer_decompressed_aggressive`: Number of buffers decompressed by queries aggressively.



`num_msg_buffer_decompressed_normal`: Number of buffers decompressed for queries.

`num_msg_buffer_decompressed_prefetch`: Number of buffers decompressed by a prefetch thread.

`num_msg_buffer_decompressed_write`: Number of buffers decompressed for writes.

`num_pivots_fetched_prefetch`: Number of pivot nodes fetched by a prefetch thread.

`num_pivots_fetched_query`: Number of pivot nodes fetched for queries.

`num_pivots_fetched_write`: Number of pivot nodes fetched for writes.

`num_basements_fetched_aggressive`: Number of basement nodes fetched from disk aggressively.

`num_basements_fetched_normal`: Number of basement nodes fetched from disk for queries.

`num_basements_fetched_prefetch`: Number of basement nodes fetched from disk by a prefetch thread.

`num_basements_fetched_write`: Number of basement nodes fetched from disk for writes.

`num_msg_buffer_fetched_aggressive`: Number of buffers fetched from disk aggressively.

`num_msg_buffer_fetched_normal`: Number of buffers fetched from disk for queries.

`num_msg_buffer_fetched_prefetch`: Number of buffers fetched from disk by a prefetch thread.

`num_msg_buffer_fetched_write`: Number of buffers fetched from disk for writes.

`dictionary inserts multi`: This is the total number of rows that have been inserted into all primary and secondary indexes combined, when those inserts have been done with a single recovery log entry for the entire row. (For example, inserting a row into a table with one primary and two secondary indexes will normally increase this count by three).

`dictionary inserts multi fail`: This is the number of multi-index insert operations that failed.

`dictionary deletes multi`: This is the total number of rows that have been deleted from all primary and secondary indexes combined, when those deletes have been done with a single recovery log entry for the entire row.

`dictionary deletes multi fail`: This is the number of multi-index delete operations that failed.

`dictionary updates multi`: This is the total number of rows that have been updated in all primary and secondary indexes combined, if those updates have been done with a single recovery log entry for the entire row.

`dictionary updates multi fail`: This is the number of multi-index update operations that failed.

`dictionary point queries`: This is the total number of point (or random) queries that have been made of all primary and secondary indexes combined.

`dictionary sequential queries`: This is the total number of sequential (range) queries that have been made of all primary and secondary indexes combined. Each row fetched by a range query will increment this counter. (For example, if a range query fetches ten rows from a table with one primary and two secondary indexes, then this counter will increase by thirty.)

`num_db_open`: Number of db_open operations.

`num_db_close`: Number of db_close operations.

`num_open_dbs`: Number of currently open dbs.



`max_open_dbs`: Max number of simultaneously open dbs.

`le_max_committed_xr`: This is the maximum number of committed transaction records that were stored on disk in a new or modified row.

`le_max_provisional_xr`: This is the maximum number of provisional transaction records that were stored on disk in a new or modified row.

`le_max_memsize`: This is the maximum number of bytes that were stored on disk as a new or modified row. This is the maximum uncompressed size of any row stored in TokuDB that was created or modified since the server started.

`le_expanded`: This is the number of times that an expanded memory mechanism was used to store a new or modified row on disk.

`ydb lock`: This shows the state of one of our internal locks, which we call the ydb lock.

`ydb lock counter`: How many times the ydb lock has been locked.

`num_waiters_now`: This is the number of client threads that are currently waiting for the ydb lock.

`max_waiters`: This is the maximum number of threads that have ever been in simultaneous contention for the ydb lock.

`total_sleep_time`: This is the total time, in microseconds, that all threads have ever slept because of contention for the ydb lock.

`max_time_ydb_lock_held`: This field is incorrectly displayed and should be ignored.

`total_time_ydb_lock_held`: This is the total time (in seconds) that the ydb lock was held.

`total_time_since_start`: This is the total time (in seconds) between the creation of the ydb lock and the most recent time it was locked or unlocked.

`cachetable lock taken`: This is the number of times that one of our internal locks, the cachetable lock, was locked.

`cachetable lock released`: This is the number of times the cachetable lock was unlocked.

`cachetable hit`: This is a count of how many times the application was able to access your data in the database's internal cache.

`cachetable miss`: This is a count of how many times the application was unable to access your data in the internal cache.

`cachetable misstime`: This is the total time, in microseconds, of how long the database has had to wait for a disk read to complete.

`cachetable wait reading`: This is the total number of times that one of our internal functions had to wait for a disk read.

`cachetable wait writing`: This is the total number of times that one of our internal functions had to wait for a disk write.

`cachetable wait checkpoint`: This is the total number of times that one of our internal functions had to wait for a disk write due to checkpoint.

`cachetable puts (new nodes)`: This is the total number of times that our memory management algorithm has created a new block of memory to be managed.

`cachetable prefetches`: This is the total number of times that a block of memory has been prefetched into the database's cache. Data is prefetched when the database's algorithms determine that a block of memory is likely to be accessed by the application.

`cachetable maybe_get_and_pins`: This is the number of times that one of our internal functions has been called.



`cachetable maybe_get_and_pin_hits`: This is the number of times that one of our internal memory management functions has had a cache hit.

`cachetable evictions`: Number of blocks evicted from cache.

`cachetable size_current`: This is a count, in bytes, of how much of your uncompressed data is currently in the database's internal cache.

`cachetable size_limit`: This is a count, in bytes, of how much of your uncompressed data will fit in the database's internal cache.

`cachetable size_max`: This is the maximum size, in bytes, that the cache has ever reached. It will normally be slightly greater than `cachetable size_limit`.

`cachetable.size_leaf`: The number of bytes of leaf nodes in the cache.

`cachetable.size_nonleaf`: The number of bytes of nonleaf nodes in the cache.

`cachetable.size_rollback`: The number of bytes of rollback nodes in the cache.

`cachetable size_writing`: This is the number of bytes that are currently queued up to be written to disk.

`max range locks`: This is the system's total limit of how many range locks are supported.

`range locks in use`: This is how many range locks are currently in use.

`memory available for range locks`: This is the system's total limit of how much memory is available for use by range locks.

`memory in use for range locks`: This is how much memory is currently in use for range locks.

`range lock escalation successes`: This is the number of times range locks have been escalated. Lock escalation is a process of coalescing multiple range locks into a single lock on a single range.

`range lock escalation failures`: This is the number of times the range lock escalation procedure could not be performed.

`range read locks acquired`: This is the number of times that a range read lock was acquired.

`range read locks unable to be acquired`: This is the number of times that a range read lock was requested but was unavailable.

`range read locks exhausted`: This is the number of times that the pool of range read locks was completely depleted.

`range write locks acquired`: This is the number of times that a range write lock was acquired.

`range write locks unable to be acquired`: This is the number of times that a range write lock was requested but was unavailable.

`range write locks exhausted`: This is the number of times that the pool of range write locks was completely depleted.

`range.lt.create`: Number of locktrees created.

`range.lt.create.fail`: Number of locktree create failures.

`range.lt.destroy`: Number of locktrees destroyed.

`range.lt.num`: Number of locktrees currently in use (should be created - destroyed).

`range.lt.num.max`: Max number of locktrees that have existed simultaneously.



`directory_read_locks`: This is the number of times that one of our internal locks, called a directory read lock, was taken.

`directory_read_locks_fail`: This is the number of times that one of our internal locks, called a directory read lock, was unable to be taken.

`directory_write_locks`: This is the number of times that one of our internal locks, called a directory write lock, was taken.

`directory_write_locks_fail`: This is the number of times that one of our internal locks, called a directory write lock, was unable to be taken.

`fsync count`: This is the total number of times the database has flushed the operating system's file buffers to disk.

`fsync time`: This the total time, in microseconds, used to fsync to disk.

`logger ilock count`: This is the number of times that one of our internal locks, the logger input lock, was locked and released.

`logger olock count`: This is the number of times that one of our internal locks, the logger output lock, was locked and released.

`logger swap count`: This is the number of times that one of our internal buffer pairs has been swapped.

`most recent disk full`: This is the most recent time when the disk file system was entirely full. If the disk has never been full, then this value will be "Dec 31, 1969" on Linux hosts.

`threads currently blocked by full disk`: This is the number of threads that are currently blocked because they are attempting to write to a full disk. This is normally zero. If this value is non-zero, then a warning will appear in the "disk free space" field.

`ENOSPC blocked count`: This is the number of times that an attempt to write to disk failed because the disk was full. If the disk is full, this number will continue increasing until space is available.

`ENOSPC reserve count (redzone)`: This is the number of database inserts that have been rejected because the amount of disk free space was less than the reserve.

`loader create (success)`: This is the number of times one of our internal objects, a loader, has been created.

`loader create fail`: This is the number of times a loader was requested but could not be created.

`loader put`: This is the total number of rows that were inserted using a loader.

`loader put.fail`: This is the total number of rows that were unable to be inserted using a loader.

`loader close (success)` : This is the number of loaders that successfully created the requested table.

`loader close fail`: This is the number of loaders that were unable to create the requested table.

`loader abort`: This is the number of loaders that were aborted.

`loaders current`: This is the number of loaders that currently exist.

`loader max`: This is the maximum number of loaders that ever existed simultaneously.

`log suppress (success)`: This is the number of times a log suppression optimization was successfully used.



`log suppress fail`: This is the number of times a log suppression optimization was requested but could not be used.

`indexer create (success)`: This is the number of times one of our internal objects, a indexer, has been created.

`indexer create fail`: This is the number of times a indexer was requested but could not be created.

`indexer build (success)`: This is the total number of times that indexes were created using a indexer.

`indexer build fail`: This is the total number of times that indexes were unable to be created using a indexer.

`indexer close (success)` : This is the number of indexers that successfully created the requested index(es).

`indexer close fail`: This is the number of indexers that were unable to create the requested index(es).

`indexer abort`: This is the number of indexers that were aborted.

`indexers current`: This is the number of indexers that currently exist.

`indexer max`: This is the maximum number of indexers that ever existed simultaneously.

`cachetable size_cachepressure`: The number of bytes causing cache pressure (the sum of buffers and workdone counters).

`checkpoint client wait on cs lock`: The number of times a client checkpoint waited for the checkpoint-safe lock.

`checkpoint client wait on mo lock`: The number of times a client thread waited for the multi-operation lock.

`checkpoint other wait on cs lock`: The number of times a non-client, non-scheduled, non-transaction checkpoint waited for the checkpoint-safe lock.

`checkpoint other wait on mo lock`: The number of times a non-client, non-scheduled, non-transaction checkpoint waited for the multi-operation lock.

`checkpoint sched wait on cs lock`: The number of times a scheduled checkpoint waited for the checkpoint-safe lock.

`checkpoint sched wait on mo lock`: The number of times a scheduled checkpoint waited for the multi-operation lock.

`checkpoint txn wait on cs lock`: The number of times a transaction commit checkpoint waited for the checkpoint-safe lock.

`checkpoint txn wait on mo lock`: The number of times a transaction commit checkpoint waited for the multi-operation lock.

`checkpoint waiters max`: This is the maximum number of threads ever simultaneously waiting for the checkpoint-safe lock to perform a checkpoint.

`checkpoint waiters now`: This is the current number of threads simultaneously waiting for the checkpoint-safe lock to perform a checkpoint.

`cleaner_num_dirtied_for_leaf_merge`: The number of nodes dirtied by the "flush from root" process to merge a leaf node.

`cleaner_num_leaf_merges_completed`: The number of times the cleaner thread successfully merges a leaf.



`cleaner_num_leaf_merges_running`: The number of cleaner thread leaf merges in progress.

`cleaner_num_leaf_merges_started`: The number of times the cleaner thread tries to merge a leaf.

`hot_max_root_flush_count`: The maximum number of flushes from the root ever required to optimize a tree.

`hot_num_aborted`: The number of HOT operations that have been aborted.

`hot_num_completed`: The number of HOT operations that have successfully completed.

`hot_num_started`: The number of HOT operations that have begun.

`malloc mmap threshold`: The threshold for malloc to use mmap.

`malloc count`: Number of calls to malloc().

`free count`: Number of calls to free().

`realloc count`: Number of calls to realloc().

`malloc fail`: Number of failed calls to malloc().

`realloc fail`: Number of failed calls to realloc().

`mem requested`: Total number of bytes requested from allocator.

`mem used`: Total number of bytes allocated by allocator.

`mem freed`: Total number of mallocated bytes freed (used - freed = bytes in use).

`max_mem_in_use`: Maximum memory footprint of the storage engine, the max value of (used - freed).

`allocator version`: Version string from in-use memory allocator.



An example of engine status:

```
mysql> show engine tokudb status;
```

Type	Name	Status
TokuDB	Version	5.1.52-tokudb-5.2.7-38674
TokuDB	disk free space	more than 10 percent of total file system space
TokuDB	time of environment creation	Thu Jan 12 19:40:01 2012
TokuDB	time of engine startup	Tue Jan 17 15:31:36 2012
TokuDB	time now	Tue Jan 17 15:31:45 2012
TokuDB	checkpoint period	60
TokuDB	checkpoint status code (0 = idle)	0
TokuDB	last checkpoint began	Tue Jan 17 15:31:36 2012
TokuDB	last complete checkpoint began	Tue Jan 17 15:31:36 2012
TokuDB	last complete checkpoint ended	Tue Jan 17 15:31:36 2012
TokuDB	last complete checkpoint LSN	42
TokuDB	checkpoints taken	1
TokuDB	checkpoints failed	0
TokuDB	checkpoint waiters now	0
TokuDB	checkpoint waiters max	0
TokuDB	checkpoint client wait on mo lock	0
TokuDB	checkpoint client wait on cs lock	0
TokuDB	checkpoint sched wait on cs lock	0
TokuDB	checkpoint client wait on cs lock	0
TokuDB	checkpoint txn wait on cs lock	0
TokuDB	checkpoint other wait on cs lock	0
TokuDB	checkpoint sched wait on mo lock	0
TokuDB	checkpoint client wait on mo lock	0
TokuDB	checkpoint txn wait on mo lock	0
TokuDB	checkpoint other wait on mo lock	0
TokuDB	cleaner period	1
TokuDB	cleaner iterations	5
TokuDB	txn begin	3
TokuDB	txn commits	3
TokuDB	txn aborts	0
TokuDB	txn close (commit+abort)	3
TokuDB	txn_num_open	0
TokuDB	txn_max_open	2
TokuDB	txn oldest live	0
TokuDB	txn oldest starttime	Wed Dec 31 19:00:00 1969
TokuDB	next LSN	51
TokuDB	dictionary inserts	0
TokuDB	dictionary inserts fail	0
TokuDB	dictionary deletes	0
TokuDB	dictionary deletes fail	0
TokuDB	dictionary updates	0
TokuDB	dictionary updates fail	0
TokuDB	dictionary broadcast updates	0
TokuDB	dictionary broadcast updates fail	0
TokuDB	leafentry updates	0
TokuDB	leafentry broadcast updates	0
TokuDB	descriptor_set	0
TokuDB	partial_fetch_hit	5
TokuDB	partial_fetch_miss	0
TokuDB	partial_fetch_compressed	0
TokuDB	partial_evictions_nonleaf	0
TokuDB	partial_evictions_leaf	0
TokuDB	msn_discards	0
TokuDB	max_workdone	0
TokuDB	total_searches	5
TokuDB	total_retries	2
TokuDB	max_search_excess_retries	0
TokuDB	max_search_root_tries	0
TokuDB	search_root_retries	0



TokuDB	search_tries_gt_height	0
TokuDB	search_tries_gt_heightplus3	0
TokuDB	cleaner_executions	9
TokuDB	cleaner_total_nodes	0
TokuDB	cleaner_hl_nodes	0
TokuDB	cleaner_hgt1_nodes	0
TokuDB	cleaner_empty_nodes	0
TokuDB	cleaner_nodes_dirtied	0
TokuDB	cleaner_max_buffer_size	0
TokuDB	cleaner_min_buffer_size	18446744073709551615
TokuDB	cleaner_total_buffer_size	0
TokuDB	cleaner_max_buffer_workdone	0
TokuDB	cleaner_min_buffer_workdone	18446744073709551615
TokuDB	cleaner_total_buffer_workdone	0
TokuDB	cleaner_num_leaf_merges_started	0
TokuDB	cleaner_num_leaf_merges_running	0
TokuDB	cleaner_num_leaf_merges_completed	0
TokuDB	cleaner_num_dirtied_for_leaf_merge	0
TokuDB	flush_total	0
TokuDB	flush_in_memory	0
TokuDB	flush_needed_io	0
TokuDB	flush_cascades	0
TokuDB	flush_cascades_1	0
TokuDB	flush_cascades_2	0
TokuDB	flush_cascades_3	0
TokuDB	flush_cascades_4	0
TokuDB	flush_cascades_5	0
TokuDB	flush_cascades_gt_5	0
TokuDB	disk_flush_leaf	0
TokuDB	disk_flush_nonleaf	0
TokuDB	disk_flush_leaf_for_checkpoint	0
TokuDB	disk_flush_nonleaf_for_checkpoint	0
TokuDB	create_leaf	0
TokuDB	create_nonleaf	0
TokuDB	destroy_leaf	0
TokuDB	destroy_nonleaf	0
TokuDB	split_leaf	0
TokuDB	split_nonleaf	0
TokuDB	merge_leaf	0
TokuDB	merge_nonleaf	0
TokuDB	dirty_leaf	0
TokuDB	dirty_nonleaf	0
TokuDB	balance_leaf	0
TokuDB	hot_num_started	0
TokuDB	hot_num_completed	0
TokuDB	hot_num_aborted	0
TokuDB	hot_max_root_flush_count	0
TokuDB	msg_bytes_in	0
TokuDB	msg_bytes_out	0
TokuDB	msg_bytes_curr	0
TokuDB	msg_bytes_max	0
TokuDB	msg_num	0
TokuDB	msg_num_broadcast	0
TokuDB	num_basements_decompressed_normal	0
TokuDB	num_basements_decompressed_aggressive	0
TokuDB	num_basements_decompressed_prefetch	0
TokuDB	num_basements_decompressed_write	0
TokuDB	num_msg_buffer_decompressed_normal	0
TokuDB	num_msg_buffer_decompressed_aggressive	0
TokuDB	num_msg_buffer_decompressed_prefetch	0
TokuDB	num_msg_buffer_decompressed_write	0
TokuDB	num_pivots_fetched_query	2
TokuDB	num_pivots_fetched_prefetch	0
TokuDB	num_pivots_fetched_write	1
TokuDB	num_basements_fetched_normal	2



TokuDB	num_basements_fetched_aggressive	0
TokuDB	num_basements_fetched_prefetch	0
TokuDB	num_basements_fetched_write	1
TokuDB	num_msg_buffer_fetched_normal	0
TokuDB	num_msg_buffer_fetched_aggressive	0
TokuDB	num_msg_buffer_fetched_prefetch	0
TokuDB	num_msg_buffer_fetched_write	0
TokuDB	dictionary inserts multi	0
TokuDB	dictionary inserts multi fail	0
TokuDB	dictionary deletes multi	0
TokuDB	dictionary deletes multi fail	0
TokuDB	dictionary updates multi	0
TokuDB	dictionary updates multi fail	0
TokuDB	dictionary point queries	5
TokuDB	dictionary sequential queries	0
TokuDB	num_db_open	1
TokuDB	num_db_close	0
TokuDB	num_open_dbs	1
TokuDB	max_open_dbs	1
TokuDB	le_max_committed_xr	0
TokuDB	le_max_provisional_xr	0
TokuDB	le_max_memsize	0
TokuDB	le_expanded	0
TokuDB	ydb lock	Unlocked
TokuDB	ydb lock counter	32
TokuDB	num_waiters_now	0
TokuDB	max_waiters	1
TokuDB	total_sleep_time	0
TokuDB	max_time_ydb_lock_held	0.008784
TokuDB	total_time_ydb_lock_held	0.009161
TokuDB	total_time_since_start	1.188098
TokuDB	cachetable lock taken	64
TokuDB	cachetable lock released	64
TokuDB	cachetable hit	5
TokuDB	cachetable miss	3
TokuDB	cachetable misstime	35
TokuDB	cachetable wait reading	0
TokuDB	cachetable wait writing	0
TokuDB	cachetable wait checkpoint	0
TokuDB	cachetable puts (new nodes)	0
TokuDB	cachetable prefetches	0
TokuDB	cachetable maybe_get_and_pins	0
TokuDB	cachetable maybe_get_and_pin_hits	0
TokuDB	cachetable size_current	945
TokuDB	cachetable size_limit	6312065024
TokuDB	cachetable size_max	1284
TokuDB	cachetable size_writing	0
TokuDB	cachetable size_nonleaf	0
TokuDB	cachetable size_leaf	945
TokuDB	cachetable size_rollback	0
TokuDB	cachetable size_cachepressure	0
TokuDB	cachetable evictions	0
TokuDB	max range locks	4294967295
TokuDB	range locks in use	0
TokuDB	memory available for range locks	789008128
TokuDB	memory in use for range locks	360
TokuDB	range lock escalation successes	0
TokuDB	range lock escalation failures	0
TokuDB	range read locks acquired	5
TokuDB	range read locks unable to be acquired	0
TokuDB	range read locks exhausted	0
TokuDB	range write locks acquired	0
TokuDB	range write locks unable to be acquired	0
TokuDB	range write locks exhausted	0
TokuDB	range_lt_create	3



TokuDB	range_lt_create_fail	0
TokuDB	range_lt_destroy	0
TokuDB	range_lt_num	3
TokuDB	range_lt_num_max	3
TokuDB	directory_read_locks	0
TokuDB	directory_read_locks_fail	0
TokuDB	directory_write_locks	0
TokuDB	directory_write_locks_fail	0
TokuDB	fsync count	2
TokuDB	fsync time	39158
TokuDB	logger ilock count	16
TokuDB	logger olock count	8
TokuDB	logger swap count	1
TokuDB	most recent disk full	Wed Dec 31 19:00:00 1969
TokuDB	threads currently blocked by full disk	0
TokuDB	ENOSPC blocked count	0
TokuDB	ENOSPC reserve count (redzone)	0
TokuDB	loader create (success)	0
TokuDB	loader create fail	0
TokuDB	loader put	0
TokuDB	loader put_fail	0
TokuDB	loader close (success)	0
TokuDB	loader close fail	0
TokuDB	loader abort	0
TokuDB	loaders current	0
TokuDB	loader max	0
TokuDB	log suppress (success)	0
TokuDB	log suppress fail	0
TokuDB	indexer create (success)	0
TokuDB	indexer create fail	0
TokuDB	indexer build (success)	0
TokuDB	indexer build fail	0
TokuDB	indexer close (success)	0
TokuDB	indexer close fail	0
TokuDB	indexer abort	0
TokuDB	indexers current	0
TokuDB	indexer max	0
TokuDB	malloc count	328
TokuDB	free count	148
TokuDB	realloc count	6
TokuDB	malloc fail	0
TokuDB	realloc fail	0
TokuDB	mem requested	37023753
TokuDB	mem used	37074920
TokuDB	mem freed	3198384
TokuDB	max mem in use	34650968
TokuDB	malloc mmap threshold	4194304
TokuDB	allocator version	2.2.5-0-gfcl1bb70e5f0d9a58b39efa39cc549b5af5104760

237 rows in set (0.00 sec)

5.6 Missing Log Files

Q: What do I do if I delete my logs files or they are otherwise missing?

A: You'll need to recover from a backup. It is essential that the log files be present in order to restart the database.

5.7 Isolation Levels

Q: What is the default isolation level for TokuDB?

A: It is repeatable-read (MVCC).

Q: How can I change the isolation level?

A: TokuDB supports repeatable-read, serializable, read-uncommitted and read-committed isolation levels (other levels are not supported). TokuDB employs pessimistic locking, and aborts a transaction when a lock conflict is detected.

To guarantee that lock conflicts do not occur, use repeatable-read, read-uncommitted or read-committed isolation level.

5.8 Lock wait timeout exceeded

Q: Why do my MySQL clients get lock timeout errors for my update queries? And what should my application do when it gets these errors?

A: Updates can get lock timeouts if some other transaction is holding a lock on the rows being updated for longer than the `tokudb_lock_timeout`. You may want to increase this timeout.

If an update deadlocks, then the transaction should abort and retry.



5.9 Query Cache

Q: Does TokuDB support the query cache?

A: Yes. This is turned on or off in the `.cnf` file. Please make sure that the size of the cache is set to something larger than 0, as this, in effect, turns the cache off.

5.10 Row Size

Q: What is the maximum row size?

A: The maximum row size is 32MiB.

5.11 NFS & CIFS

Q: Can the data directories reside on a disk that is NFS or CIFS mounted?

A: No, NFS- and CIFS-mounted disks are not supported. Please make sure that disks are directly attached or SAN-attached.

5.12 Using other Storage Engines

Q: Can the MyISAM and InnoDB Storage Engines be used?

A: MyISAM and InnoDB can be used directly in conjunction with TokuDB. Please note that you should not overcommit memory between InnoDB and TokuDB. The total memory assigned to both caches must be less than physical memory.

5.13 Using MariaDB Patches with TokuDB

Q: Can I use MariaDB source code patches with TokuDB?

A: Yes, but you need to apply Tokutek patches as well as your patches to MariaDB 5.2.3 to build a binary that works with the Tokutek Fractal Tree library. Tokutek provides a patched version of 5.2.3 as well. Tokutek's patches have all been released under the GPLv2 – contact us at support@tokutek.com for detailed instructions.

5.14 Truncate Table vs Delete from Table

Q: Which is faster, `truncate table` or `delete from table`?

A: Please use `truncate table` whenever possible. A table truncation runs in constant time, whereas a `delete from table` requires a row-by-row deletion and thus runs in time linear in the table size.



5.15 Foreign Keys

Q: Does TokuDB enforce foreign key constraints?

A: No, TokuDB ignores foreign key declarations.

5.16 Dropping Indexes

Q: Is dropping an index in TokuDB hot?

A: No, the table is locked for the amount of time it takes the file system to delete the file associated with the index.

5.17 Submitting a Bug Report

Q: How do I submit a bug report?

A: Please submit bug reports by email to support@tokutek.com. We respond to all bug reports within one business day.

If you encounter a bug, please submit as much of the following information as possible.

- The exact query and the resulting output, if applicable.
- The `mysql` error file (which can be found in `$hostname.err` in the MariaDB data directory).
- The output of `show variables`.
- A description of your hardware (how many cores, what processor, what kind of disks).
- A description of your operating system and distribution (e.g., is it Centos 5.2? 32-bit or 64-bit?)
- The output of `df $MYSQLDATADIR`
- The output of `show engine tokudb status;`
- A directory listing of the data directories, e.g., by running this command:

```
ls -lR $MYSQLDATADIR
```
- Schema definitions.
- A tarball containing the database files (if you feel comfortable letting us have the file, and it isn't too big to fit in an email).

- A core file if MariaDB crashed.

One easy way to turn on core files is to add the line

```
core-file
```

to the `[mysqld]` section of your `.cnf` file. If you want to use config files from standard locations (e.g. `/etc/my.cnf`) and don't want to modify them, create a new file named `enable_core_files.cnf` containing:

```
[mysqld]
core-file
```

and use the following flag with `mysqld` or `mysqld_safe`:

```
--defaults-extra-file=enable_core_files.cnf
```

6 Compiling MariaDB from source



It is not necessary to build MariaDB and the TokuDB for MariaDB[®] handler from source, but if you want to this section tells you how to do so and how to link with the Tokutek[®] Fractal Tree[®] Library.

Please note that Tokutek made some changes to the MariaDB source that are required to either fix bugs or aid in performance, so if you are compiling from source you must use the Tokutek version of MariaDB that is based on the MariaDB 5.2.3 source.

The instructions in this section have been written with the assumption that you know what you are doing and are familiar with building MariaDB.

After executing these instructions, follow the instructions in the Tokutek Quick Start Guide to install and start the server.

6.1 System and Hardware Requirements

Operating Systems: These instructions were tested using CentOS 5.5. They are expected to work with other Linux distributions.

Compiler: These instructions were tested using gcc 4.1.2 and gcc-c++ 4.1.2.

Packages: You will need the following packages installed at the noted revision or later:

- bison 2.3
- ccache 2.4
- ncurses-devel 5.6
- readline 5.2
- autoconf 2.61
- automake 1.10
- libtool 1.5
- libevent 1.4

Processor Architecture: TokuDB runs on the following architectures.

- X86_64

Please send email to support@tokutek.com if you are interested in other architectures.

Disk space: 1GB.

6.2 Download and verify files

Please download and verify these files from the Tokutek web site and put them in the build directory.

```
mariadb-5.2.3-tokudb-5.2.7-38674-src.tar.gz
mariadb-5.2.3-tokudb-5.2.7-38674-src.tar.gz.md5
tokufractalreeindex-5.2.7-38674-linux-x86_64.tar.gz
tokufractalreeindex-5.2.7-38674-linux-x86_64.tar.gz.md5
```

6.3 Configure and build

NOTE: The C/C++ compilers default as gcc44 and g++44. You can override these by creating the environment variables CC and CXX or by modifying the tokudb.build.bash script directly on lines 16 and 17.



Execute the following:

```
# extract the tokudb.build.bash script from the source tarball
tar xzf mariadb-5.2.3-tokudb-5.2.7-38674-src.tar.gz

# move the script to the current directory
mv mariadb-5.2.3-tokudb-5.2.7-38674-src/scripts/tokudb.build.bash .

# make the script executable
chmod 744 tokudb.build.bash

# build the binary release tarball
./tokudb.build.bash
```

6.4 Install

Once the source has been compiled, you can install it by using the newly created tarball in the source directory and following the instructions in the Quick Start Guide on installing and starting the server with one exception, you must copy the following files from tokufractalreeindex-5.2.7-38674-linux-x86_64.tar.gz to the lib/mysql folder after installing:

- lib/libtokufractalreeindex-5.2.7-38674.so
- lib/libtokuportability-5.2.7-38674.so

7 Additional Questions and Support

Tokutek looks forward to your feedback on this product and we encourage you to contact us by e-mail at support@tokutek.com. Our Technical Services team monitors this email address to ensure prompt and personal service.

The TokuDB User Guide and Frequently Asked Questions (FAQ) are available on our website at <http://www.tokutek.com/resources/product-docs>.

Our Evaluation Guide will help you understand all the capabilities of TokuDB and how they apply to your specific use-case, it can be downloaded at <http://www.tokutek.com/resources/tokudb-eval-guide>.

Visit us at <http://www.tokutek.com/support> for more information.



Appendix

A Known Issues

InnoDB: Our binary includes the InnoDB plugin that ships with MariaDB. Other versions of the InnoDB plugin can be recompiled using our modified MariaDB source and used in conjunction with TokuDB. Please note, however, that the InnoDB plugin is not supported by TokuDB customer service.

Replication and binary logging: TokuDB supports binary logging and replication, with one restriction. TokuDB currently does not implement a lock on the auto-increment function, so concurrent insert statements with one or more of the statements inserting multiple rows will result in a non-deterministic interleaving of the auto-increment values. When running replication with concurrent inserts, the auto-increment fields in the slave table may not match the auto-increment values in the master table.

See: <http://askmonty.org/wiki/Manual:Contents> for additional details on auto-increment and replication.

In addition, when using the command `REPLACE INTO` or `INSERT IGNORE` on tables with no secondary indexes or tables where secondary indexes are subsets of the primary, the session variable `tokudb_pk_insert_mode` controls whether row based replication will work. Please see section 4.1.4.

Show Table Status: Some fields in `SHOW TABLE STATUS` and `SHOW INDEX` display 0 or NULL. For example:

```
show table status \G
***** 1. row *****
      Name: tokudb_table
      Engine: TokuDB
      Version: 10
      Row_format: Fixed
      Rows: 687000
      Avg_row_length: 33
      Data_length: 22671000
      Max_data_length: 9223372036854775807
      Index_length: 39684132
      Data_free: 0
      Auto_increment: 688001
      Create_time: NULL
      Update_time: NULL
      Check_time: NULL
      Collation: latin1_swedish_ci
      Checksum: NULL
      Create_options:
      Comment:
1 row in set (0.25 sec)
```

This will be fixed in a future release.

Uninformative error message: `load data infile` can sometimes produce the error message `ERROR 1030 (HY000): Got error 1 from storage engine`. The message should say that the error is caused by insufficient disk space for the temporary files created by the loader. If this happens, turn on the `tokudb_load_save_space` session variable and try again. See section 4.1.5.



Changing the auto-increment value: The command `alter table foo auto_increment=new_val` does not change the auto increment value to `new_val`. It leaves the auto increment value unchanged.

B Release History

Version 5.2.7 includes important improvements:

- Performance enhancement for multi-client workloads.
- Improved point-query performance.
- Ability to disable prefetching for range queries with LIMIT via the `tokudb.disable_prefetching_session` variable.
- Reduced memory footprint.
- Improved CPU utilization on bulk loads.
- Improved range query performance.
- Increased maximum row size from 4MiB to 32MiB.
- Hot Column Rename is now supported.
- Hot Optimize Table is now supported. Note that Optimize Table does not rebuild indexes in TokuDB, since TokuDB indexes do not fragment. Instead, it flushes pending work induced by column additions and deletions.

Version 5.0.6 included:

- Features:
 - Added table create time and last update time to `information_schema.tables` and `SHOW TABLE STATUS`
- Bug fixes:
 - Fixed a bug in 5.0.5 that could, under certain circumstances, cause a crash.

Version 5.0.5 included:

- Features:
 - `SELECT FOR UPDATE` is now supported.
 - Changed to single file download.
 - Created script to assist users building from source.
 - Point update deadlocks are avoided since TokuDB uses write locks for these operations.
 - Replace into deadlocks are avoided since TokuDB uses write locks for these operations.
 - Added more information to engine status output.

Version 5.0.4 included:

- Performance:
 - Opening and closing very large tables is much faster than in 5.0.3.
 - Adding blob and text columns is much faster than in 5.0.3.



- Bug fixes:
 - Fixed a bug in 5.0.3 that would cause a crash when creating a hot index that is a clustering key.
 - Fixed a bug in 5.0.3 that could, under certain circumstances, cause a crash when adding a column. (For more information, see <http://bugs.mysql.com/bug.php?id=61017>.)
 - Fixed a bug in 5.0.3 that could, under certain circumstances, cause a crash when updating a table that has a blob.
 - Fixed a bug in 5.0.3 that could, under rare circumstances, cause the creation of an incorrect index when a hot index was created simultaneously with a `replace into` operation.
 - Fixed a bug in 5.0.3 that would cause `SHOW TABLE STATUS` to show an incorrect number of rows for tables with more than two billion rows.

Version 5.0.3 included:

- Features:
 - Hot Schema Changes:
 - * Hot index creation is now supported. When an index is created, the database remains available (online) for all normal operations, including queries, inserts, deletes, and updates. Once the index creation completes, it becomes immediately available for future queries. Any inserts/deletes/updates to the table during index creation are incorporated in the new index. This allows adding of indexes without having to suspend normal operation of other database applications.
 - * Hot Column Addition/Deletion is now supported. After a brief interruption, the database remains available (online) for all normal operations, including queries, inserts, deletes, and updates during an alter table that adds or deletes columns. There is a small (seconds to a few minutes) time during which the table is locked because MySQL requires that tables be closed and reopened during an alter table.
 - Snapshot isolation is now the default isolation level in TokuDB. TokuDB implements snapshot isolation with multi-version concurrency control (MVCC).
 - Automatic upgrade of databases created by TokuDB versions 4.2 or later is now supported. It is not necessary to dump and load databases running under TokuDB 4.2 or later.
 - Row locking is improved. Row locks are now limited solely by the memory allocated, not by their use per index and not by their total number.
 - The `tokudb_tmp_dir` server variable is added.
 - Show Engine Status is improved.
 - Size of uncompressed user data is now available via the information schema. Commands
 - * `show engine tokudb user_data;`
 - * `show engine tokudb user_data_exact;`
 are now deprecated. The alternatives are, respectively:
 - * `select * from information_schema.TokuDB_user_data;`
 - * `select * from information_schema.TokuDB_user_data_exact;`

Version 4.2.0 included:

- Fixed a bug in previous 4.x versions. The bug affects tables built using the bulk loader available in TokuDB versions 4.x which can cause inconsistencies in the tables and possibly the loss of some data. We recommend that all users use TokuDB version 4.2.0 or later to rebuild tables that were built using earlier versions of the bulk loader.



Version 4.1.1 included:

- Bug fixes:
 - Fixed a bug in 4.1.0.

Version 4.1.0 included:

- Features:
 - `SAVEPOINT` is now supported.
 - Progress reporting of bulk loads is improved. Progress is now displayed as a percentage.
- Performance:
 - TokuDB 4.1.1 uses less virtual memory than TokuDB 4.0.0 when loading an empty table from a data file.
 - TokuDB 4.1.1 loads an empty table from a data file faster than Tokudb 4.0.0.
- Bug fixes:
 - Fixed a bug where the server, while testing for clustering keys, would crash when running a join query.
 - Fixed a bug that caused a crash in rare cases when simultaneously loading multiple empty tables from data files.
 - Fixed a bug that caused a crash when multiple instances of TokuDB were started using a common directory.

Version 4.0.0 included:

- Performance
 - Bulk Loading: TokuDB now optimizes for the case of creating a table or adding an index.
 - Point queries: Point queries are now faster.
 - `INSERT IGNORE` is now faster in many cases.
- Logging Optimizations
 - The logs needed for ACID transactions are now considerably smaller. Furthermore, when a transaction commits, the space associated with logging for that transaction is reclaimed more quickly. Finally, smaller logs means faster recovery in many cases.
- Feature
 - `READ COMMITTED` isolation level.

Version 3.1.0 included:

- Feature
 - Improved handling of a full disk: When disk free space is below a configurable reserve TokuDB disallows insertions so that more disk space can be made available or `mysqld` can be shut down cleanly without triggering a recovery on restart. Also, if the disk becomes completely full TokuDB will freeze, allowing the user to free some disk space and allowing the database to resume operation.



- Performance
 - Faster group commits.
 - Faster crash recovery.
- Improved diagnostics.
 - `SHOW ENGINE STATUS` has been improved to include information on free disk space and some new internal diagnostics.
 - `SHOW PROCESSLIST` shows the progress of commits and aborts.
- Bug fixes:
 - Fixed bugs which sometimes caused recovery to require manual intervention. These include bugs related to truncated log files, e.g. when file system fills; recovery on empty log files; and recovery when there are files with names similar to log files. Recovery now works automatically, with no human intervention, in all these cases.
 - Fixed a bug related to chars in non-default collations. Chars may now be used in non-default collations.
 - Added session variable `tokudb_pk_insert_mode` to handle interaction of `REPLACE INTO` commands with triggers and row based replication.



Version 3.0.4 included:

- Upgraded from MySQL 5.1.36 to 5.1.43.
- This release supports two versions of MySQL, 5.1.43 and 5.5.1-m2.
- This release adds a new client session variable called `tokudb_write_lock_wait` so the user can control the amount of time a thread waits for write locks to be attained before timing out.
- This release adds support for *group commit*.

Version 3.0.3 included:

- This release supports two versions of MySQL, 5.1.36 and 5.5.1-m2.

Version 3.0.2 included:

- This release fixes an issue in the recovery code that may prevent full recovery following a crash. There is no work-around, and users testing crash recovery should upgrade immediately.

Version 3.0.1 included:

- This release fixes a bug, in which certain statements failed when run in the `READ UNCOMMITTED` isolation level. The statements include `REPLACE INTO`, `INSERT IGNORE`, and `INSERT ... ON DUPLICATE KEY UPDATE`.

Version 3.0.0 included:

- Full ACID-compliant transactions, including support for transactional statements such as `BEGIN TRANSACTION`, `END TRANSACTION`, `COMMIT` and `ROLLBACK`.
- New command `show engine tokudb user_data` returns the amount of data the user in the system.
- Bug fixes.

Version 2.2.0 included:

- Increase in multithreaded performance through the use of a fair scheduler for queries and inserts.
- Added support for command `show engine tokudb status`.
- Provide more detailed progress indicators in `show processlist`.
- A script is provided to show how much tokudb user data is in the system. TokuDB pricing is based on this script.
- Faster bulk load into empty table in these scenarios:
 1. `load data infile ...`
 2. `insert into TABLE_NAME select * ...`

Version 2.1.0 included:

- Support for InnoDB.
- Upgraded from MySQL 5.1.30 to 5.1.36.
- Faster indexing of sequential keys.
- Faster bulk loads on tables with auto-increment fields.
- Faster range queries in some circumstances.

Version 2.0.2 included:

- Blob support: the maximum row size has been increased from 80KiB to 4MiB.
- Performance Improvement: updates on tables with clustering indexes have become up to twice as fast.

Version 2.0.1 included:

- Crash safety for Windows.
- `CREATE INDEX` is now abortable. An abort will typically take less than 2 minutes.
- Fixed a problem with “replace into” or “insert into ... on duplicate key” on tables with blob/text fields that could cause a crash or an incorrect insertion under certain circumstances.

Version 2.0.0 included:

- Crash safety for Linux: TokuDB for MySQL now includes checkpointing so that after a power failure or a crash, each table comes up in a consistent state.



C 3rd Party Libraries

C.1 jemalloc

Copyright (C) 2002-2011 Jason Evans <jasone@canonware.com>.

All rights reserved.

Copyright (C) 2007-2010 Mozilla Foundation. All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the above copyright notice(s), this list of conditions and the following disclaimer.
2. Redistributions in binary form must reproduce the above copyright notice(s), this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDER(S) "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER(S) BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

Copyright (C) 2009-2011 Facebook, Inc.

All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

- * Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
- * Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
- * Neither the name of Facebook, Inc. nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

