PERCONA

www.percona.com

Percona TokuDB Documentation

Release 7.5.7

July 10, 2015

TokuDB® is a highly scalable, zero-maintenance downtime MySQL storage engine that delivers indexing-based query acceleration, improved replication performance, unparalleled compression, and live schema modification. TokuDB is a drop-in storage engine requiring no changes to MySQL or MariaDB applications or code and is fully ACID and MVCC compliant.

Additional features unique to TokuDB include:

- Up to 25x Data Compression
- Fast Inserts
- Eliminates Slave Lag with *Read Free Replication*
- 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, expansion, and rename - TokuDB tables support insertions, deletions and queries without down-time when an alter table adds, deletes, expands, or renames columns
- On-line Backup (Enterprise Edition)

For more information on installing and using TokuDB for MySQL and MariaDB, click on the following links:

CHAPTER

ONE

USING TOKUDB

Caution: Do not move or modify any TokuDB files. You will break the database, and need to recover the database from a backup.

1.1 Fast Insertions and 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 MySQL and TokuDB. Here are some resources to get you started:

- "Understanding Indexing" by Zardosht Kasheff (video) (slides)
- Rule of Thumb for Choosing Column Order in Indexes
- TokuView Blog, the Tokutek Blog has many entries on indexing. Please see these specific examples and more:
  - Covering Indexes: Orders-of-Magnitude Improvements
  - Introducing Multiple Clustering Indexes
  - Clustering Indexes vs. Covering Indexes
  - How Clustering Indexes Sometimes Helps UPDATE and DELETE Performance
- *High Performance MySQL, 3rd Edition* by Baron Schwartz, Peter Zaitsev, Vadim Tkachenko, Copyright 2012, O'Reilly Media. See Chapter 5, *Indexing for High Performance*.

1.2 Clustering Secondary Indexes

One of the keys to exploiting TokuDB's strength in indexing is to make use of clustering secondary indexes.

To define a secondary index as clustering, simply add the word `CLUSTERING` before the key definition. For example:

```
CREATE TABLE table (
  column_a INT,
  column_b INT,
  column_c INT,
  PRIMARY KEY index_a (column_a),
  CLUSTERING KEY index_b (column_b)) ENGINE = TokuDB;
```

In the previous example, the primary table is indexed on *column_a*. Additionally, there is a secondary clustering index (named *index_b*) 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 following query will run very fast using the clustering *index_b*:

```
SELECT column_c
  FROM table
  WHERE column_b BETWEEN 10 AND 100;
```

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 information about using clustering indexes, see Introducing Multiple Clustering Indexes

1.3 Hot Index Creation

TokuDB enables 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. More information is available in *TokuDB Variables*.

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

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

mysql> CREATE INDEX index ON table (field_name);
```

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.

1.4 Hot Column Add, Delete, Expand, and Rename (HCADER)

TokuDB enables you to add or delete columns in an existing table, expand char, varchar, varbinary, and integer type columns in an existing table, or rename an existing column in a table with little blocking of other updates and queries. HCADER typically blocks other queries with a table lock for no more than a few seconds. After that initial short-term table locking, the system modifies each row (when adding, deleting, or expanding 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 HCADER, observe the following guidelines:

- The work of altering the table for column addition, deletion, or expansion is performed as subsequent operations touch parts of the Fractal Tree, both in the primary index and secondary indexes.

  You can force the column addition, deletion, or expansion work to be performed all at once using the standard syntax of `OPTIMIZE TABLE X`, when a column has been added to, deleted from, or expanded in table X. It is important to note that as of TokuDB version 7.1.0, `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, deletion, or expansion.

- Each hot column addition, deletion, or expansion operation must be performed individually (with its own SQL statement). If you want to add, delete, or expand multiple columns use multiple statements.
- Avoid adding, deleting, or expanding 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 HCADER is dominated by the time it takes to flush dirty pages, because MySQL 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.
- Hot column expansion operations are only supported to char, varchar, varbinary, and integer data types. Hot column expansion is not supported if the given column is part of the primary key or any secondary keys.
- Rename only one column per statement. Renaming more than one column will revert to the standard MySQL blocking behavior. The proper syntax is as follows:

```
ALTER TABLE table
  CHANGE column_old column_new
  DATA_TYPE REQUIRED_NESS DEFAULT
```

Here's an example of how that might look:

```
ALTER TABLE table
  CHANGE column_old column_new
  INT(10) NOT NULL;
```

Notice that all of the column attributes must be specified. `ALTER TABLE table CHANGE column_old column_new;` induces a slow, blocking column rename.

- Hot column rename does not support the following data types: `TIME`, `ENUM`, `BLOB`, `TINYBLOB`, `MEDIUMBLOB`, `LONGBLOB`. Renaming columns of these types will revert to the standard MySQL blocking behavior.
- Temporary tables cannot take advantage of HCADER. Temporary tables are typically small anyway, so altering them using the standard method is usually fast.

1.5 Compression Details

TokuDB offers different levels of compression, which trade off between the amount of CPU used and the compression achieved. Standard compression uses less CPU but generally compresses at a lower level, high compression uses more CPU and generally compresses at a higher level. We have seen compression up to 25x on customer data.

Compression in TokuDB occurs on background threads, which means that high compression need not slow down your database. Indeed, in some settings, we've seen higher overall database performance with high compression.

Note: We recommend that users use standard compression on machines with six or fewer cores, and high compression on machines with more than six cores.

The ultimate choice depends on the particulars of how a database is used, and we recommend that users use the default settings unless they have profiled their system with high compression in place.

Compression is set on a per-table basis and is controlled by setting row format during a `CREATE TABLE` or `ALTER TABLE`. For example:

```
CREATE TABLE table (
  column_a INT NOT NULL PRIMARY KEY,
  column_b INT NOT NULL) ENGINE=TokuDB
  ROW_FORMAT=row_format;
```

If no row format is specified in a `CREATE TABLE`, the table is compressed using whichever row format is specified in the session variable `tokudb_row_format`. If no row format is set nor is `tokudb_row_format`, the zlib compressor is used.

`row_format` and `tokudb_row_format` variables accept the following values:

- `tokudb_default`: This sets the compression to the default behavior. As of TokuDB 7.1.0, the default behavior is to compress using the zlib library. In the future this behavior may change.
- `tokudb_fast`: This sets the compression to use the quicklz library.
- `tokudb_small`: This sets the compression to use the lzma library.

In addition, you can choose a compression library directly, which will override previous values. The following libraries are available:

- `tokudb_zlib`: Compress using the zlib library, which provides mid-range compression and CPU utilization.
- `tokudb_quicklz`: Compress using the quicklz library, which provides light compression and low CPU utilization.
- `tokudb_lzma`: Compress using the lzma library, which provides the highest compression and high CPU utilization.
- `tokudb_uncompressed`: This setting turns off compression and is useful for tables with data that cannot be compressed.

1.6 Changing Compression of a Table

Modify the compression used on a particular table with the following command:

```
ALTER TABLE table
  ROW_FORMAT=row_format;
```

Note: Changing the compression of a table only affects newly written data (dirtied blocks). After changing a table's compression you can run `OPTIMZE TABLE` to rewrite all blocks of the table and its indexes.

1.7 Read Free Replication

TokuDB slaves can be configured to perform significantly less read IO in order to apply changes from the master. By utilizing the power of Fractal Tree indexes:

- insert/update/delete operations can be configured to eliminate read-modify-write behavior and simply inject messages into the appropriate Fractal Tree indexes
- update/delete operations can be configured to eliminate the IO required for uniqueness checking

To enable Read Free Replication, the servers must be configured as follows:

- On the replication master:

  - Enable row based replication: set `BINLOG_FORMAT=ROW`

- On the replication slave(s):
  - The slave must be in read-only mode: set `read_only=1`
  - Disable unique checks: set `tokudb_rpl_unique_checks=0`
  - Disable lookups (read-modify-write): set `tokudb_rpl_lookup_rows=0`

Note: You can modify one or both behaviors on the slave(s).

Note: In MySQL 5.5 and MariaDB 5,5, only tables with a defined primary key are eligible for this optimization. The limitation does not apply to MySQL 5.6, Percona Server 5.6, and MariaDB 10.

Note: As long as the master is using row based replication, this optimization is available on a TokuDB slave. This means that it's available even if the master is using InnoDB or MyISAM tables, or running non-TokuDB binaries.

1.8 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 every 60 seconds, and this specifies the time from the beginning of one checkpoint to the beginning of the next. If a checkpoint requires more than the defined checkpoint period to complete, the next checkpoint begins immediately. 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 the `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.

1.9 Managing Log Size

TokuDB keeps log files back to the most recent checkpoint. Whenever a log file reaches 100 MB, 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 60 seconds 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.

1.10 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.

1.11 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 MySQL. 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, the following command should disable the write cache:

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

There are some cases when 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 information, see the XFS FAQ.

In the following cases, you must disable the write cache:

- If you use the ext3 filesystem
- If you use LVM (although recent Linux kernels, such as Fedora 12, have fixed this problem)
- If you use Linux's software RAID
- If you use a RAID controller with battery-backed-up memory. This may seem counter-intuitive. For more information, see the XFS FAQ

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

1.12 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 estimation 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.

1.13 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 SELECT Syntax.

Note: Creating this file does not save the schema of your table, so you may want to create a copy of that as well.

1.14 Hot Backup

Hot Backup enables the database to be backed up with no downtime. The Tokutek hot backup library intercepts system calls that write files and duplicates the writes to the backup directory. An in depth design discussion of Hot Backup is available at:

- TokuDB Hot Backup - Part 1
- TokuDB Hot Backup - Part 2

1.14.1 Hot Backup 7.5.5 and later

Configuration

Before using the Hot Backup plugin, the plugin must be installed. To install it, execute the following command:

```
mysql> install plugin tokudb_backup soname 'tokudb_backup.so';
```

Once the plugin is installed, it is then possible to execute a backup. The destination directory where the backups will be located must be empty, otherwise a failure will occur. To back up a database, the user sets the `tokudb_backup_dir` variable to an empty directory as follows:

```
mysql> set tokudb_backup_dir='/path_to_empty_directory';
```

As soon as the variable is set, the backup will begin.

Monitoring Progress and Checking Errors

Hot backup updates the *processlist* state while the backup is in progress. Users will be able to see the output by running `show processlist` or `show full processlist`.

There are two variables that can be used to capture errors from Hot Backup. They are `@@tokudb_backup_last_error` and `@@tokudb_backup_last_error_string`. When Hot Backup encounters an error, these will report on the error number and the error string respectively. For example, the following output shows these parameters following an attempted backup to a directory that was not empty:

```
mysql> set tokudb_backup_dir='/tmp/backupdir';
ERROR 1231 (42000): Variable 'tokudb_backup_dir' can't be set to the value of '/tmp/backupdir'

mysql> select @@tokudb_backup_last_error;
+----------------------------+
| @@tokudb_backup_last_error |
+----------------------------+
|                         17 |
+----------------------------+

mysql> @@tokudb_backup_last_error_string;
ERROR 1064 (42000): You have an error in your SQL syntax; check the manual that corresponds to your M

mysql> select @@tokudb_backup_last_error_string;
+-------------------------------------------------+
| @@tokudb_backup_last_error_string               |
+-------------------------------------------------+
| tokudb backup couldn't create needed directories. |
+-------------------------------------------------+
```

Optional Settings

tokudb_backup_allowed_prefix This system-level variable restricts the location of the destination directory where the backups can be located. Attempts to backup to a location outside of the directory this variable points to or its children will result in an error. The default is null, backups have no restricted locations. This read-only variable can be set in the `my.cnf` config file and displayed with the `show variables` command.

```
mysql> show variables where variable_name='tokudb_backup_allowed_prefix';
+------------------------------+-----------+
| Variable_name                | Value     |
+------------------------------+-----------+
| tokudb_backup_allowed_prefix | /dumpdir  |
+------------------------------+-----------+
```

tokudb_backup_throttle This session-level variable throttles the write rate in bytes per second of the backup to prevent Hot Backup from crowding out other jobs in the system. The default and max value is 18446744073709551615.

```
mysql> set tokudb_backup_throttle=1000000;
```

tokudb_backup_dir When set, this session-level variable serves two purposes, to point to the destination directory where the backups will be dumped and to kick off the backup as soon as it's set.

tokudb_backup_last_error This session variable contains the error number from the last backup. 0 indicates success.

tokudb_backup_last_error_string This session variable contains the error string from the last backup.

1.14.2 Hot Backup Prior to 7.5.5

There is no requirement to install a plugin prior to 7.5.5. Hot Backup is compiled into the executable. To run Hot Backup, the destination directory must exist, be writable and empty. Once this directory is created, the backup can be run using the following command:

```
mysql> backup to '/path_to_empty_directory';
```

CHAPTER

TWO

GETTING STARTED

2.1 System and Hardware Requirements

Operating Systems: TokuDB is currently supported on 64-bit Linux only.

Memory: TokuDB Requires at least 1GB of main memory but for best results, we recommend to run with at least 2GB of main memory.

Disk space and configuration: Please make sure to allocate enough disk space for data, indexes and logs. In our users' experience, TokuDB achieves up to 25x space savings on data and indexes over InnoDB due to high compression.

A virtual machine image is available for evaluations on Windows and Mac, please contact us at support@percona.com for more information.

2.2 Creating Tables and Loading Data

2.2.1 Creating TokuDB Tables

TokuDB tables are created the same way as other tables in MySQL by specifying `engine=TokuDB` in the table definition. For example, the following command creates a table with a single column and uses the TokuDB storage engine to store its data:

```
CREATE TABLE table (
  id INT(11) NOT NULL) ENGINE=TokuDB;
```

2.2.2 Loading Data

Once TokuDB tables have been created, data can be inserted or loaded using standard MySQL insert or bulk load operations. For example, the following command loads data from a file into the table:

```
LOAD DATA INFILE file
  INTO TABLE table;
```

Note: For more information about loading data, see the MySQL 5.5 reference manual.

2.2.3 Migrating Data from an Existing Database

Use the following command to convert an existing table for the TokuDB storage engine:

```
ALTER TABLE table
  ENGINE=TokuDB;
```

2.3 Bulk Loading Data

The TokuDB bulk loader imports data much faster than regular MySQL with InnoDB. To make use of the loader you need flat files in either comma separated or tab separated format. The MySQL `LOAD DATA INFILE ...` statement will invoke the bulk loader if the table is empty. Keep in mind that while this is the most convenient and, in most cases, the fastest way to initialize a TokuDB table, it may not be replication safe if applied to the master

For more information, see the MySQL 5.5 Reference Manual: LOAD DATA INFILE.

To obtain the logical backup and then bulk load into TokuDB, follow these steps:

1. Create a logical backup of the original table. The easiest way to achieve this is using `SELECT ... INTO OUTFILE`. Keep in mind that the file will be created on the server.

   ```
   SELECT * FROM table
     INTO OUTFILE 'file.csv';
   ```

2. The output file should either be copied to the destination server or the client machine from which you plan to load it.

3. To load the data into the server use `LOAD DATA INFILE`. If loading from a machine other than the server use the keyword `LOCAL` to point to the file on local machine. Keep in mind that you will need enough disk space on the temporary directory on the server since the local file will be copied onto the server by the MySQL client utility.

   ```
   LOAD DATA [LOCAL] INFILE 'file.csv';
   ```

It is possible to create the CSV file using either `mysqldump` or the `mysql` client utility as well, in which case the resulting file will reside on a local directory. In these 2 cases you have to make sure to use the correct command line options to create a file compatible with `LOAD DATA INFILE`.

The bulk loader will use more space than normal for logs and temporary files while running, make sure that your file system has enough disk space to process your load. As a rule of thumb, it should be approximately 1.5 times the size of the raw data.

Note: Please read the original MySQL documentation to understand the needed privileges and replication issues needed around `LOAD DATA INFILE`.

2.4 Considerations to Run TokuDB in Production

In most cases, the default options should be left in-place to run TokuDB, however it is a good idea to review some of the configuration parameters.

2.4.1 Memory allocation

TokuDB will allocate 50% of the installed RAM for its own cache (global variable `tokudb_cache_size`). While this is optimal in most situations, there are cases where it may lead to memory over allocation. If the system tries to allocate more memory than is available, the machine will begin swapping and run much slower than normal.

It is necessary to set the `tokudb_cache_size` to a value other than the default in the following cases:

- **Running other memory heavy processes on the same server as TokuDB**: In many cases, the database process needs to share the system with other server processes like additional database instances, http server, application server, e-mail server, monitoring systems and others. In order to properly configure TokuDB's memory consumption, it's important to understand how much free memory will be left and assign a sensible value for TokuDB. There is no fixed rule, but a conservative choice would be 50% of available RAM while all the other processes are running. If the result is under 2 GB, you should consider moving some of the other processes to a different system or using a dedicated database server.

  `tokudb_cache_size` is a static variable, so it needs to be set before starting the server and cannot be changed while the server is running. For example, to set up TokuDB's cache to 4G, add the following line to your `my.cnf` file:

  ```
  tokudb_cache_size = 4G
  ```

- **System using InnoDB and TokuDB**: When using both the TokuDB and InnoDB storage engines, you need to manage the cache size for each. For example, on a server with 16 GB of RAM you could use the following values in your configuration file:

  ```
  innodb_buffer_pool_size = 2G
  tokudb_cache_size = 8G
  ```

- **Using TokuDB with Federated or FederatedX tables**: The Federated engine in MySQL and FederatedX in MariaDB allow you to connect to a table on a remote server and query it as if it were a local table (please see the MySQL documentation: 14.11. The FEDERATED Storage Engine for details). When accessing the remote table, these engines could import the complete table contents to the local server to execute a query. In this case, you will have to make sure that there is enough free memory on the server to handle these remote tables. For example, if your remote table is 8 GB in size, the server has to have more than 8 GB of free RAM to process queries against that table without going into swapping or causing a kernel panic and crash the MySQL process. There are no parameters to limit the amount of memory that the Federated or FederatedX engine will allocate while importing the remote dataset.

2.4.2 Specifying the Location for Files

As with InnoDB, it is possible to specify different locations than the default for TokuDB's data, log and temporary files. This way you may distribute the load and control the disk space. The following variables control file location:

- `tokudb_data_dir`: This variable defines the directory where the TokuDB tables are stored. The default location for TokuDB's data files is the MySQL data directory.
- `tokudb_log_dir`: This variable defines the directory where the TokuDB log files are stored. The default location for TokuDB's log files is the MySQL data directory. Configuring a separate log directory is somewhat involved and should be done only if absolutely necessary. We recommend to keep the data and log files under the same directory.
- `tokudb_tmp_dir`: This variable defines the directory 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 TokuDB's temporary files is the MySQL data directory.

2.4.3 Table Maintenance

Overview

The fractal tree provides fast performance by inserting small messages in the buffers in the fractal trees instead of requiring a potential IO for an update on every row in the table as required by a B-tree. Additional background

information on how fractal trees operate can be found here. For tables whose workload pattern is a high number of sequential deletes, it may be beneficial to flush these delete messages down to the basement nodes in order to allow for faster access. The way to perform this operation is via the `optimize` command.

The following extensions to the `optimize` command have been added in TokuDB version 7.5.5:

- **Hot Optimize Throttling**

  By default, table optimization will run with all available resources. To limit the amount of resources, it is possible to limit the speed of table optimization. The `tokudb_optimize_throttle` session variable determines an upper bound on how many fractal tree leaf nodes per second are optimized. The default is 0 (no upper bound) with a valid range of [0,1000000]. For example, to limit the table optimization to 1 leaf node per second, use the following setting:

  ```
  set tokudb_optimize_throttle=1;
  ```

- **Optimize a Single Index of a Table**

  To optimize a single index in a table, the `tokudb_optimize_index_name` session variable can be set to select the index by name. For example, to optimize the primary key of a table:

  ```
  set tokudb_optimize_index_name='primary';
  optimize table t;
  ```

- **Optimize a Subset of a Fractal Tree Index**

  For patterns where the left side of the tree has many deletions (a common pattern with increasing id or date values), it may be useful to delete a percentage of the tree. In this case, it is possible to optimize a subset of a fractal tree starting at the left side. The `tokudb_optimize_index_fraction` session variable controls the size of the sub tree. Valid values are in the range [0.0,1.0] with default 1.0 (optimize the whole tree). For example, to optimize the leftmost 10% of the primary key:

  ```
  set tokudb_optimize_index_name='primary';
  set tokudb_optimize_index_fraction=0.1;
  optimize table t;
  ```

CHAPTER

THREE

INSTALLATION

Tokutek provides binary versions of MySQL and MariaDB with a few minor changes to support TokuDB. The TokuDB, MyISAM and InnoDB storage engines are included in the distribution files.

The installation of MySQL from a tarball release file is described in the MySQL 5.5 reference manual. The following instructions are based on it with information specific to TokuDB.

3.1 Downloading

To get started, download the appropriate binaries from the TokuDB Download Page. The file name should follow this convention:

```
[mysql|mariadb]-[mysql|mariadb version]-tokudb-[tokudb version & build]-linux-x86_64.tar.gz.
```

For the purposes of this document, we'll use the name of the tarball corresponding to MySQL 5.5.30 with TokuDB v7.1.0. Make sure you use the name of the tarball corresponding to the file you downloaded.

After downloading, optionally verify the MD5 checksum by comparing the results of the following command to the MD5 checksum on the support page:

```
$ md5sum mysql-5.5.30-tokudb-7.1.0-linux-x86_64.tar.gz
```

At this point, it is a good idea to determine the type of installation you wish to perform. The options are:

- *Installing TokuDB on a new server with no previous MySQL installation*
- *Installing TokuDB to replace MySQL*
- *Upgrading from a previous TokuDB version*
- *Local or user space installation, optionally using MySQL Sandbox*

3.2 Installing TokuDB from Scratch

All the operations have to be performed with root privileges because you will need to perform privileged operations. Please check with your specific Linux distribution the appropriate method to perform privileged operations.

1. Create a MySQL specific group. For CentOS distributions, the GID is set to 27. Other Linux distributions may vary, even across instances of the same distribution. It is a good idea to set a uniform GID across all servers. We will use GID 927 for the rest of this document.

   ```
   $ groupadd -g 927 mysql
   ```

2. Create a MySQL system user. Similar to GID, the UID in CentOS distributions is set to 27. The user needs to be defined as a *system user*, which implies that it won't have a login shell and home directory. As with the GID, we will use UID 927 for the rest of this document.

   ```
   $ useradd -r -u 927 -g mysql mysql
   ```

3. Create the installation base directory. You may specify a different sub-directory tree to follow your installations standard.

   ```
   $ mkdir -pv /opt/tokutek
   ```

4. Extract the tarball you downloaded from our website under the base directory. You may use other directories (for example, `/usr/local` or a private directory).

   ```
   $ cd /opt/tokutek
   $ tar xvzf /path/to/mysql-5.5.30-tokudb-7.1.0-linux-x86_64.tar.gz
   ```

5. Create a symbolic link pointing to the newly created directory. This will simplify future upgrades and installations by redefining the symbolic link to a new directory.

   ```
   $ ln -sv mysql-5.5.30-tokudb-7.1.0-linux-x86_64 mysql
   ```

6. Change the ownership of the new directory and it's contents to `mysql:mysql`.

   ```
   $ cd mysql
   $ chown -Rv mysql:mysql
   ```

7. Create a default configuration file. You will have to edit this file as needed. You may leave most of the default memory allocation values as-is to allow TokuDB to use as much RAM as possible.

   ```
   $ cp -v support-files/my-small.cnf /etc/my.cnf
   ```

8. Add default data and binary directories, and mysqld user. Edit the `/etc/my.cnf` file to specify the database directory, the location for the binaries and the user to use for the mysqld process. These values are going to be needed to make sure the next step succeeds. Add the following lines in the `[mysqld]` section of the file replacing the actual values as needed:

   ```
   datadir = /var/lib/mysql
   basedir = /opt/tokutek/mysql
   user = mysql
   ```

9. You might need to edit the resulting `/etc/my.cnf` further to accommodate for other non-default settings. Create the system tables by running the following script. If needed, you may specify the options `basedir` and `datadir` with the proper paths before the `--user` option:

   ```
   $ scripts/mysql_install_db --user=mysql
   ```

10. Create a link in `/etc/init.d` to start TokuDB's server daemon as a service.

    First, edit the `mysql.server` file and make sure that `basedir` is set to `/opt/tokutek/mysql` (or the directory under which you installed the tarball). This is not needed if the variables are set up in the `/etc/my.cnf` file.

    Then, create a symbolic link under `/etc/init.d` pointing to the `mysql.server` script:

    ```
    $ ln -sv /opt/tokutek/mysql/support-files/mysql.server /etc/init.d/mysql
    ```

11. Start MySQL service.

    ```
    $ service mysql start
    ```

You may want to modify your `PATH` environment variable to include the sub-directory with the client utilities: `/opt/tokutek/mysql/bin`.

Please refer to *Verify the Installation* to confirm that everything is running as expected.

3.3 Upgrading From Previous Releases of TokuDB

TokuDB v7.1.0 supports automatic upgrades from TokuDB versions 4.1, 5.x, 6.x, and 7.x using the same configuration and data files from the older versions.

It is highly recommended that you have a full working backup of your data before upgrading any MySQL version. The backup should include the MySQL configuration files as well.

All the operations have to be performed with root privileges because you will need to perform privileged operations. Please check with your specific Linux distribution the appropriate method to perform privileged operations.

1. Create the installation base directory, unless it already exists. You may specify a different subdirectory tree to follow your installations standard.

   ```
   $ mkdir -pv /opt/tokutek
   ```

2. Extract the tarball you downloaded from our website under the base directory.

   ```
   cd /opt/tokutek
   tar xvzf /path/to/mysql-5.5.30-tokudb-7.1.0-linux-x86_64.tar.gz
   ```

3. Shutdown the existing instance. Make sure that it was a clean shutdown. It is important to make sure there are no pending transactions in the log files prior to the upgrade to avoid any potential data corruption.

   ```
   $ mysqladmin shutdown
   ```

4. Create a symbolic link pointing to the newly created directory.

   ```
   $ ln -sv mysql-5.5.30-tokudb-7.1.0-linux-x86_64 mysql
   ```

5. Create a link in `/etc/init.d` to start TokuDB's server daemon as a service.

   First, edit the `mysql.server` file and make sure that `basedir` is set to `/opt/tokutek/mysql` (or the directory under which you installed the tarball). This is not needed if the variables are set up in the `/etc/my.cnf` file.

   Then, create a symbolic link under `/etc/init.d` pointing to the `mysql.server` script:

   ```
   $ ln -sv /opt/tokutek/mysql/support-files/mysql.server /etc/init.d/mysql
   ```

6. Start the new MySQL instance. Make sure there were no errors after the startup completed by inspecting the error log.

   ```
   $ service mysql start
   ```

7. Run **mysql_upgrade** to update the system tables with any changes since the last release. It is important that you run the **mysql_upgrade** utility provided with our binaries.

   ```
   $ /opt/tokutek/mysql/bin/mysql_upgrade
   ```

8. Update the plug-ins. TokuDB v7.1 introduced 3 new plug-ins and removed 2 existing ones. Log in to MySQL and execute the following commands if upgrading a TokuDB database that was prior to v7.1 before the upgrade:

```
INSTALL PLUGIN tokudb_trx SONAME 'ha_tokudb.so';
INSTALL PLUGIN tokudb_locksSONAME 'ha_tokudb.so';
INSTALL PLUGIN tokudb_lock_waitsSONAME 'ha_tokudb.so';
DELETE FROM mysql.plugin WHERE NAME LIKE 'tokudb_user_data%';
```

Please refer to *Verify the Installation* to confirm that everything is running as expected. You should also verify the application data.

3.4 Replace an Existing MySQL Instance

The procedure to replace an existing MySQL instance should be the same as *Upgrading From Previous Releases of TokuDB*. After converting your tables to TokuDB, you should comment out the global options related to memory caches (`innodb_buffer_pool%`).

Note: When upgrading from MySQL 5.1 using InnoDB plug-in to MySQL 5.5 or MariaDB, you need to comment out the lines specifying the InnoDB plug-in libraries in your `my.cnf` file. To verify whether or not the options are enabled, you can use the following command:

```
$ my_print_defaults mysqld | grep "ha_inno"
```

The previous command should return no results for TokuDB to start cleanly.

Note: As of TokuDB version 6.1.0, it is set as the default storage engine. When migrating from an existing MySQL or MariaDB installation, you need to specify the default storage engine to InnoDB or MyISAM before the first run. Set one of the following options in the `[mysqld]` section of the `my.cnf` file:

- `default-storage-engine=MyISAM`
- `default-storage-engine=InnoDB`

This setting can be removed once the installation has been completed.

After running the `mysql_upgrade` script, run the following SQL commands to setup the TokuDB plug-in. Use the **mysql** utility. You will need to login into MySQL with a user with sufficient privileges to run the `install plugin` command:

```
INSTALL PLUGIN TokuDB SONAME 'ha_tokudb.so';
INSTALL PLUGIN tokudb_file_map SONAME 'ha_tokudb.so';
INSTALL PLUGIN tokudb_fractal_tree_info SONAME 'ha_tokudb.so';
INSTALL PLUGIN tokudb_fractal_tree_block_map SONAME 'ha_tokudb.so';
INSTALL PLUGIN tokudb_trx SONAME 'ha_tokudb.so';
INSTALL PLUGIN tokudb_locks SONAME 'ha_tokudb.so';
INSTALL PLUGIN tokudb_lock_waits SONAME 'ha_tokudb.so';
SET GLOBAL default_storage_engine=TokuDB;
```

Please refer to *Verify the Installation* to confirm that everything is running as expected. You should also verify the application data.

3.5 Local or User Space Installation

It is possible to run TokuDB as any regular user. To do so, you can extract the tarball in a private directory and run the commands from there. You have to make sure that a number of conditions are met to avoid having permission issues:

- No other MySQL instance should be running on the machine. If there is another one in place, you have to make sure your configuration will use a different port, log and data files, and configuration files to avoid conflicts.
- Make sure that you set up the log and data directories to a path where your user has full read and write access
- Read the MySQL Documentation section on running multiple instances for full details on how to specify the different options.
- The variable `tokudb_cache_size` is set to 50% of physical RAM on the server. When running multiple instances of MySQL, you may need to adjust it to a more sensible value to avoid memory over allocation, which may cause the server to become unresponsive or crash different processes, including MySQL instances.

If you decide to use this option, consider using MySQL Sandbox. MySQL Sandbox offers a number of tools to install one or more MySQL instances as a regular user taking care of all configuration details automatically. Each instance can be run independently or it can set up different replication topologies as well. As a result of the installation process, the end user will have a number of scripts to start, stop and connect to each individual instance without the need to be aware of the configuration and runtime details.

The procedure to install TokuDB for MySQL Sandbox is no different than regular MySQL. You can read the instructions to create a single sandbox here.

3.6 Installing TokuDB for MariaDB on Ubuntu Server

TokuDB for MariaDB has an unmatched dependency on Ubuntu Server v12.04. Our MariaDB build is dependent on libevent v1.4, Ubuntu Servers install libevent v2.0 by default.

When trying to start the MySQL service without installing the proper library, it will fail with the following error in the log file:

```
/opt/tokutek/mysql/bin/mysqld:
error while loading shared libraries: libevent-1.4.so.2:
cannot open shared object file: No such file or directory
```

To verify which version you have in place you may use the following command:

```
$ dpkg-query -l libevent-*

| Status=Not/Inst/Conf-files/Unpacked/halF-conf/Half-inst/trig-aWait/Trig-pend
|/ Err?=(none)/Reinst-required (Status,Err: uppercase=bad)
||/ Name                          Version                      Description
+++-=============================-============================-==================================
ii  libevent-1.4-2                1.4.14b-stable-0ubuntu1      asynchronous event notification library
ii  libevent-2.0-5                2.0.16-stable-1 Asynchronous event notification library
```

If the line for libevent-1.4 is not in the list, you may install it by issuing the following command:

```
$ sudo apt-get install libevent-1.4
```

3.7 Verify the Installation

Start a mysql client session to verify the TokuDB storage engine installation.

```
$ /opt/tokutek/mysql/bin/mysql
```

The default socket for mysqld is `/tmp/mysql.sock`. At the command prompt, execute the following command:

```
SHOW PLUGINS;
```

The output should include the following lines:

```
| TokuDB                        | ACTIVE | STORAGE ENGINE     | ha_tokudb.so ...
| TokuDB_file_map               | ACTIVE | INFORMATION SCHEMA | ha_tokudb.so ...
| TokuDB_fractal_tree_info      | ACTIVE | INFORMATION SCHEMA | ha_tokudb.so ...
| TokuDB_fractal_tree_block_map | ACTIVE | INFORMATION SCHEMA | ha_tokudb.so ...
| TokuDB_trx                    | ACTIVE | INFORMATION SCHEMA | ha_tokudb.so ...
| TokuDB_locks                  | ACTIVE | INFORMATION SCHEMA | ha_tokudb.so ...
| TokuDB_lock_waits             | ACTIVE | INFORMATION SCHEMA | ha_tokudb.so ...
```

Also execute the following command:

```
SHOW ENGINES;
```

The output should include the following line:

```
| TokuDB | YES | Tokutek TokuDB Storage Engine
```

CHAPTER

FOUR

FREQUENTLY ASKED QUESTIONS

This section contains frequently asked questions regarding TokuDB and related software. If you don't see a question that addresses your concerns, please email us.

4.1 Transactional Operations

What transactional operations does TokuDB support?

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

4.2 TokuDB and the File System

How can I determine which files belong to the various tables and indexes in my schemas?

The `tokudb_file_map` plugin lists all Fractal Tree Indexes and their corresponding data files. The `internal_file_name` is the actual file name (in the data folder).

```
mysql> SELECT * FROM information_schema.tokudb_file_map;

+-------------------------+-----------------------------------------+---------------+-------------+---
| dictionary_name         | internal_file_name                      | table_schema  | table_name  | ta
+-------------------------+-----------------------------------------+---------------+-------------+---
```

```
| ./test/tmc-key-idx_col2  | ./_test_tmc_key_idx_col2_a_14.tokudb  | test          | tmc          | ke
| ./test/tmc-main          | ./_test_tmc_main_9_14.tokudb          | test          | tmc          | ma
| ./test/tmc-status        | ./_test_tmc_status_8_14.tokudb        | test          | tmc          | st
+--------------------------+---------------------------------------+---------------+--------------+---
```

4.3 Full Disks

What happens when the disk system fills up?

The disk system may fill up during bulk load operations, such as `LOAD DATA IN FILE` 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 `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 (for more information, see *tokudb_tmp_dir*).

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 about the disk system:

- 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.

- 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 following command:

  ```
  SHOW ENGINE TokuDB STATUS;
  ```

  Make 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 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.

- In order to make space available on this system you can:
  - Add some disk space to the filesystem.
  - 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:

- 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.
- 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.
- 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.
- The `tokudb_fs_reserve_percent` variablecan not be changed once the system has started. It can only be set in `my.cnf` or on the mysqld command line.

4.4 Backup

How do I back up a system with TokuDB tables?

Enterprise Edition

The Enterprise Edition of TokuDB is capable of performing online backups. To perform a backup, execute `backup to '/path/to/backup';`. This will create backup of the server and return when complete. The backup can be used by another server using a copy of the binaries on the source server. You can view the progress of the backup by executing `show processlist;`. TokuDB enterprise backup produces a copy of your running MySQL server that is consistent at the end time of the backup process. The thread copying files from source to destination can be throttled by setting the `tokudb_backup_throttle` server variable.

The following conditions apply:

- Currently, enterprise backup only supports tables using the TokuDB storage engine and the MyISAM tables in the mysql database. Full support for InnoDB and MyISAM tables will be added in the future.

  Warning: You must disable InnoDB asynchronous IO if backing up InnoDB tables via our enterprise backup functionality. The appropriate setting is `innodb_use_native_aio=0`.

- Transactional storage engines (TokuDB and InnoDB) will perform recovery on the backup copy of the database when it is first started.
- Tables using non-transactional storage engines (MyISAM) are not locked during the copy and may report issues when starting up the backup. It is best to avoid operations that modify these tables at the end of a hot backup operation (adding/changing users, stored procedures, etc.).
- The database is copied locally to the path specified in `/path/to/backup`. This folder must exist, be writable, be empty, and contain enough space for a full copy of the database.
- Enterprise backup always makes a backup of the the MySQL `datadir` and optionally the `tokudb_data_dir`, `tokudb_log_dir`, and the binary log folder. The latter three are only

backed up separately if they are not the same as or contained in the MySQL `datadir`. None of these three folders can be a parent of the MySQL `datadir`.

- A folder is created in the given backup destination for each of the source folders.
- No other directory structures are supported. All InnoDB, MyISAM, and other storage engine files must be within the MySQL `datadir`.
- Enterprise backup does not follow symbolic links.

Community Edition

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 MySQL 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 or `mysqldump` application may also be used to get a logical backup of the database.

References

The MySQL 5.5 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, Copyright 2008, O'Reilly Media.

Cold Backup

When MySQL is shut down, a copy of the MySQL 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 MySQL data directory with all of other MySQL files. One merely has to back up this directory.

Hot Backup using mylvmbackup

The `mylvmbackup` utility, located on Launchpad, works with TokuDB. It does all of the magic required to get consistent copies of all of the MySQL 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 5.5 reference manual for details.

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

4.5 Missing Log Files

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

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

4.6 Isolation Levels

What is the default isolation level for TokuDB?

It is repeatable-read (MVCC).

How can I change the isolation level?

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.

4.7 Lock Wait Timeout Exceeded

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

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 the this timeout.

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

For more information on diagnosing locking issues, see *Lock Visualization in TokuDB*.

4.8 Query Cache

Does TokuDB support the query cache?

Yes, you can enable the query cache in the `my.cnf` file. Please make sure that the size of the cache is set to something larger than 0, as this, in effect, disables the cache.

4.9 Row Size

What is the maximum row size?

The maximum row size is 32 MiB.

4.10 NFS & CIFS

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

Yes, we do have customers in production with NFS & CIFS volumes today. However, both of these disk types can pose a challenge to performance and data integrity due to their complexity. If you're seeking performance, the switching infrastructure and protocols of a traditional network were not conceptualized for low response times and can be very difficult to troubleshoot. If you're concerned with data integrity, the possible data caching at the NFS level can cause inconsistencies between the logs and data files that may never be detected in the event of a crash. If you are thinking of using a NFS or CIFS mount, we would recommend that you use synchronous mount options, which are available from the NFS mount man page, but these settings may decrease performance. For further discussion please look here.

4.11 Using Other Storage Engines

Can the MyISAM and InnoDB Storage Engines be used?

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.

Can the Federated Storage Engines be used?

The Federated Storage Engine can also be used, however it is disabled by default in MySQL. It can be enabled by either running mysqld with `--federated` as a command line parameter, or by putting `federated` in the `[mysqld]` section of the `my.cnf` file.

For more information see the MySQL 5.5 Reference Manual: FEDERATED Storage Engine.

4.12 Using MySQL Patches with TokuDB

Can I use MySQL source code patches with TokuDB?

Yes, but you need to apply Tokutek patches as well as your patches to MySQL to build a binary that works with the Tokutek Fractal Tree library. Tokutek provides a patched version of MySQL as well.

4.13 Truncate Table vs Delete from Table

Which is faster, TRUNCATE TABLE or DELETE FROM TABLE?

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 to the table size.

4.14 Foreign Keys

Does TokuDB enforce foreign key constraints?

No, TokuDB ignores foreign key declarations.

4.15 Dropping Indexes

Is dropping an index in TokuDB hot?

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

CHAPTER

FIVE

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.

Contents

5.1 Client Session Variables

`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=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 disabled 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.

`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, disable the `tokudb_commit_sync` session variable as follows:

```
SET tokudb_commit_sync=OFF;
```

Disabling this variable may make the system run faster. However, transactions committed since the last checkpoint are not guaranteed to survive a crash.

`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 `(column_a INT, column_b INT, column_c INT, PRIMARY KEY (column_a,column_b), KEY (column_b))` is affected, because the only column in the key of *column_b* is present in the primary key. TokuDB 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.

`tokudb_load_save_space`

This session variable changes the behavior of the bulk loader. When it is disabled the bulk loader stores intermediate data using uncompressed files (which consumes additional CPU), whereas on compresses the intermediate files. It is enabled by default.

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 disable `tokudb_prelock_empty` and try again.

More information is available in **:ref:‘Known Issues <known-issues>‘_**.

`tokudb_prelock_empty`

By default, in 7.1.0, 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. Disabling `tokudb_prelock_empty` optimizes for this multi-transaction case by turning off preemptive prelocking.

Note: If this variable is set to off, fast bulk loading is turned off as well.

`tokudb_create_index_online`

This variable controls whether indexes created with the `CREATE INDEX` command are hot (if enabled), or offline (if disabled). 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.

Note: Hot index creation is slower than offline index creation.

By default, `tokudb_create_index_online` is enabled.

`tokudb_disable_slow_alter`

This variable controls whether slow alter tables are allowed. For example, the following command is slow because HCADER does not allow a mixture of column additions, deletions, or expansions:

```
ALTER TABLE table
  ADD COLUMN column_a INT,
  DROP COLUMN column_b;
```

By default, `tokudb_disable_slow_alter` is disabled, and the engine reports back to mysql that this is unsupported resulting in the following output:

```
ERROR 1112 (42000): Table 'test_slow' uses an extension that doesn't exist in this MySQL version
```

`tokudb_block_size`

Fractal tree internal and leaf nodes default to 4,194,304 bytes (4 MB). The session variable `tokudb_block_size` controls the target uncompressed size of these nodes.

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

`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 and the default is 65,536 (64 KB). A smaller value favors read performance for point and small range scans over large range scans and higher compression. 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 for an existing table without a dump and reload.

`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 (128 KB).

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.

`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.

`tokudb_row_format`

This session variable controls the default compression algorithm used to compress data when no row format is specified in the `CREATE TABLE` command. See **:ref:'Compression Details <compress-details>'_**.

`tokudb_analyze_time`

This session variable controls the number of seconds an analyze operation will spend on each index when calculating cardinality. Cardinality is shown by executing

```
SELECT INDEXES FROM table_name;
```

If an analyze is never performed on a table then the cardinality is 1 for primary key indexes and unique secondary indexes, and NULL (unknown) for all other indexes. Proper cardinality can lead to improved performance of complex SQL statements. The default value is 5.

`tokudb_lock_timeout_debug`

The following values are available:

0 No lock timeouts or lock deadlocks are reported.

1 A JSON document that describes the lock conflict is stored in the `tokudb_last_lock_timeout` session variable

2 A JSON document that describes the lock conflict is printed to the MySQL error log.

Supported since 7.5.5: In addition to the JSON document describing the lock conflict, the following lines are printed to the MySQL error log:

- A line containing the blocked thread id and blocked sql
- A line containing the blocking thread id and the blocking sql.

3 A JSON document that describes the lock conflict is stored in the `tokudb_last_lock_timeout` session variable and is printed to the MySQL error log.

Supported since 7.5.5: In addition to the JSON document describing the lock conflict, the following lines are printed to the MySQL error log:

- A line containing the blocked thread id and blocked sql
- A line containing the blocking thread id and the blocking sql.

`tokudb_last_lock_timeout`

This session variable contains a JSON document that describes the last lock conflict seen by the current MySQL client. It gets set when a blocked lock request times out or a lock deadlock is detected.

The `tokudb_lock_timeout_debug` session variable must have bit 0 set for this behavior, otherwise this session variable will be empty.

`tokudb_bulk_fetch`

This session variable determines if our bulk fetch algorithm is used for `SELECT` and `DELETE` statements. `SELECT` statements include pure `SELECT ...` statements, as well as `INSERT INTO table-name ... SELECT ...`, `CREATE TABLE table-name ... SELECT ...`, `REPLACE INTO table-name ... SELECT ...`, `INSERT IGNORE INTO table-name ... SELECT ...`, and `INSERT INTO table-name ... SELECT ... ON DUPLICATE KEY UPDATE`.

By default, `tokudb_bulk_fetch` is enabled.

`tokudb_support_xa`

This session variable defines whether or not the prepare phase of an XA transaction performs an `fsync()`.

By default, `tokudb_support_xa` is enabled.

`tokudb_optimize_throttling`

Supported since 7.5.5

By default, table optimization will run with all available resources. To limit the amount of resources, it is possible to limit the speed of table optimization. The `tokudb_optimize_throttling` session variable determines an upper bound on how many fractal tree leaf nodes per second are optimized. The default is 0 (no upper bound) with a valid range of [0,1000000].

`tokudb_optimize_index_name`

Supported since 7.5.5

To optimize a single index in a table, the `tokudb_optimize_index_name` session variable can be enabled to select the index by name.

`tokudb_optimize_index_fraction`

Supported since 7.5.5

For patterns where the left side of the tree has many deletions (a common pattern with increasing id or date values), it may be useful to delete a percentage of the tree. In this case, it's possible to optimize a subset of a fractal tree starting at the left side. The `tokudb_optimize_index_fraction` session variable controls the size of the sub tree. Valid values are in the range [0.0,1.0] with default 1.0 (optimize the whole tree).

`tokudb_backup_throttle`

This session level variable throttles the write rate in bytes per second of the backup to prevent Hot Backup from crowding out other jobs in the system. The default and max values are 18446744073709551615

`tokudb_backup_dir`

Supported since 7.5.5

When enabled, this session level variable serves two purposes, to point to the destination directory where the backups will be dumped and to kick off the backup as soon as it is set.

`tokudb_backup_last_error`

Supported since 7.5.5

This session variable will contain the error number from the last backup. 0 indicates success.

`tokudb_backup_last_error_string`

Supported since 7.5.5

This session variable will contain the error string from the last backup.

5.2 MySQL Server Variables

`tokudb_loader_memory_size`

Limits the amount of memory that the TokuDB bulk loader will use for each loader instance, defaults to 100 MB. Increasing this value may provide a performance benefit when loading extremely large tables with several secondary indexes.

Note: Memory allocated to a loader is taken from the TokuDB cache, defined as `tokudb_cache_size`, and may impact the running workload's performance as existing cached data must be ejected for the loader to begin.

`tokudb_fsync_log_period`

Controls the frequency, in milliseconds, for `fsync()` operations. If set to 0 then the `fsync()` behavior is only controlled by the tokudb commit sync, which is on or off. The default values is 0.

`tokudb_cache_size`

This variable configures the size in bytes of the TokuDB cache table. The default cache table size is 1/2 of physical memory. Tokutek highly recommends using the default setting if using buffered IO, if using direct IO then consider setting this parameter to 80% of available memory.

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.

`tokudb_directio`

When enabled, TokuDB employs Direct IO rather than Buffered IO for writes. When using Direct IO, consider increasing `tokudb_cache_size` from its default of 1/2 physical memory.

By default, `tokudb_directio` is disabled.

`tokudb_lock_timeout`

This variable 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 gets a `deadlock found` error (-30995), then you need to abort the current transaction and retry it.

`tokudb_data_dir`

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

`tokudb_log_dir`

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

`tokudb_tmp_dir`

This variable specifies the directory 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 MySQL data directory.

`tokudb_checkpointing_period`

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

`tokudb_write_status_frequency`, `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 updated 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.

`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 *Full Disks* for more information.

`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.

`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.

`tokudb_backup_throttle`

(Enterprise Edition) This variable specifies the maximum number of bytes per second the copier of a hot backup process will consume. Lowering its value will cause the hot backup operation to take more time but consume less IO on the server. The default value is 18446744073709551615.

`tokudb_rpl_lookup_rows`

When disabled, TokuDB replication slaves skip row lookups for *delete row* log events and *update row* log events, which eliminates all associated read IO for these operations.

Note: Optimization is only enabled when `read_only` is 1 `binlog_format` is ROW.

By default, `tokudb_rpl_lookup_rows` is enabled.

`tokudb_rpl_lookup_rows_delay`

This server variable allows for simulation of long disk reads by sleeping for the given number of microseconds prior to the row lookup query, it should only be set to a non-zero value for testing.

By default, `tokudb_rpl_lookup_rows_delay` is disabled.

`tokudb_rpl_unique_checks`

When disabled, TokuDB replication slaves skip uniqueness checks on inserts and updates, which eliminates all associated read IO for these operations.

Note: Optimization is only enabled when `read_only` is 1, *binlog_format* is ROW.

By default, `tokudb_rpl_unique_checks` is enabled.

`tokudb_rpl_unique_checks_delay`

This server variable allows for simulation of long disk reads by sleeping for the given number of microseconds prior to the row lookup query, it should only be set to a non-zero value for testing.

By default, `tokudb_rpl_unique_checks_delay` is disabled.

`tokudb-backup-plugin-version`

Supported since 7.5.5:

This server variable documents the version of the hot backup plugin

`tokudb_backup_version`

Supported since 7.5.5:

This server variable documents the version of the hot backup library.

`tokudb_backup_allowed_prefix`

Supported since 7.5.5:

This system-level variable restricts the location of the destination directory where the backups can be located. Attempts to backup to a location outside of the directory this variable points to or its children will result in an error.

The default is null, backups have no restricted locations. This read only variable can be set in the `my.cnf` file and displayed with the `show variables` command.

`tokudb_rpl_check_readonly`

Supported since 7.5.5:

The TokuDB replication code will run row events from the binlog with RFR when the slave is in read only mode. The `tokudb_rpl_check_readonly` variable is used to disable the slave read only check in the TokuDB replication code.

This allows RFR to run when the slave is NOT read only. By default, `tokudb_rpl_check_readonly` is enabled (check slave read only). Do NOT change this value unless you completely understand the implications!

CHAPTER

SIX

TROUBLESHOOTING

6.1 Known Issues

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

Replication and binary logging: TokuDB supports binary logging and replication, with one restriction. TokuDB does not implement a lock on the auto-increment function, so concurrent insert statements with one or more of the statements inserting multiple rows may result in a non-deterministic interleaving of the auto-increment values. When running replication with these concurrent inserts, the auto-increment values on the slave table may not match the auto-increment values on the master table. Note that this is only an issue with Statement Based Replication (SBR), and not Row Based Replication (RBR).

For more information about auto-increment and replication, see the MySQL Reference Manual: AUTO_INCREMENT handling in InnoDB.

In addition, when using the `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.

Uninformative error message: The `load data infile` command can sometimes produce `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.

Transparent Huge Pages: TokuDB will refuse to start if transparent huge pages are enabled. Transparent huge page support can be disabled by issuing the following as root:

```
# echo never > /sys/kernel/mm/redhat_transparent_hugepage/enabled
```

Note: The previous command needs to be executed after every reboot, because it defaults to `always`.

Limitations of MySQL 5.1 and MariaDB 5.2 Versions: There are features present in the MySQL 5.5 and MariaDB 5.5 versions of TokuDB that are not available in the MySQL 5.1 and MariaDB 5.2 versions of TokuDB, including the following:

- Internal locking mechanism may prevent hot operations from starting and cause them to eventually time out if the system is under insert/update/delete load on the requested table.
- Partitioned tables can use the TokuDB storage engine, but schema change operations are not performed hot. Index creation, column adds, drops, and expansions, and compression changes are all blocking operations on partitioned tables.
- Altering column default operations is not performed hot, it is a blocking operation.
- Column expansion of char, varchar, varbinary, and integer types are not performed hot, they are blocking operations.

XA behavior vs. InnoDB: InnoDB forces a deadlocked XA transaction to abort, TokuDB does not.

MySQL 5.5 and MariaDB 5.5 does not support hot combined adding/dropping of indexes: Multiple indexes may be added hot or dropped hot, but the operations cannot be combined in a single DDL statement.

6.2 Lock Visualization in TokuDB

TokuDB uses key range locks to implement serializable transactions, which are acquired as the transaction progresses. The locks are released when the transaction commits or aborts (this implements two phase locking).

TokuDB stores these locks in a data structure called the lock tree. The lock tree stores the set of range locks granted to each transaction. In addition, the lock tree stores the set of locks that are not granted due to a conflict with locks granted to some other transaction. When these other transactions are retired, these pending lock requests are retried. If a pending lock request is not granted before the lock timer expires, then the lock request is aborted.

Lock visualization in TokuDB exposes the state of the lock tree with tables in the information schema. We also provide a mechanism that may be used by a database client to retrieve details about lock conflicts that it encountered while executing a transaction.

6.2.1 The `tokudb_trx` table

The `tokudb_trx` table in the information schema maps TokuDB transaction identifiers to MySQL client identifiers. This mapping allows one to associate a TokuDB transaction with a MySQL client operation.

The following query returns the MySQL clients that have a live TokuDB transaction:

```
SELECT * FROM information_schema.tokudb_trx,
  information_schema.processlist
  WHERE trx_mysql_thread_id = id;
```

6.2.2 The `tokudb_locks` table

The `tokudb_locks` table in the information schema contains the set of locks granted to TokuDB transactions.

The following query returns all of the locks granted to some TokuDB transaction:

```
SELECT * FROM information_schema.tokudb_locks;
```

The following query returns the locks granted to some MySQL client:

```
SELECT id FROM information_schema.tokudb_locks,
  information_schema.processlist
  WHERE locks_mysql_thread_id = id;
```

6.2.3 The `tokudb_lock_waits` table

The `tokudb_lock_waits` table in the information schema contains the set of lock requests that are not granted due to a lock conflict with some other transaction.

The following query returns the locks that are waiting to be granted due to a lock conflict with some other transaction:

```
SELECT * FROM information_schema.tokudb_lock_waits;
```

6.2.4 The `tokudb_lock_timeout_debug` session variable

The `tokudb_lock_timeout_debug` session variable controls how lock timeouts and lock deadlocks seen by the database client are reported.

The following values are available:

0 No lock timeouts or lock deadlocks are reported.

1 A JSON document that describes the lock conflict is stored in the `tokudb_last_lock_timeout` session variable

2 A JSON document that describes the lock conflict is printed to the MySQL error log.

Supported since 7.5.5: In addition to the JSON document describing the lock conflict, the following lines are printed to the MySQL error log:

- A line containing the blocked thread id and blocked sql
- A line containing the blocking thread id and the blocking sql.

3 A JSON document that describes the lock conflict is stored in the `tokudb_last_lock_timeout` session variable and is printed to the MySQL error log.

Supported since 7.5.5: In addition to the JSON document describing the lock conflict, the following lines are printed to the MySQL error log:

- A line containing the blocked thread id and blocked sql
- A line containing the blocking thread id and the blocking sql.

6.2.5 The `tokudb_last_lock_timeout` session variable

The `tokudb_last_lock_timeout` session variable contains a JSON document that describes the last lock conflict seen by the current MySQL client. It gets set when a blocked lock request times out or a lock deadlock is detected. The `tokudb_lock_timeout_debug` session variable should have bit 0 set (decimal 1).

6.2.6 Example

Suppose that we create a table with a single column that is the primary key.

```
mysql> SHOW CREATE TABLE table;

Create Table: CREATE TABLE `table` (
`id` int(11) NOT NULL,
PRIMARY KEY (`id`)) ENGINE=TokuDB DEFAULT CHARSET=latin1
```

Suppose that we have 2 MySQL clients with ID's 1 and 2 respectively. Suppose that MySQL client 1 inserts some values into `table`. TokuDB transaction 51 is created for the insert statement. Since autocommit is disabled, transaction 51 is still live after the insert statement completes, and we can query the `tokudb_locks` table in information schema to see the locks that are held by the transaction.

```
mysql> SET autocommit=OFF;
mysql> INSERT INTO table VALUES (1),(10),(100);

Query OK, 3 rows affected (0.00 sec)
Records: 3  Duplicates: 0  Warnings: 0

mysql> SELECT * FROM information_schema.tokudb_locks;

+--------------+-----------------------+---------------+----------------+-----------------+----------
| locks_trx_id | locks_mysql_thread_id | locks_dname   | locks_key_left | locks_key_right | locks_tab
+--------------+-----------------------+---------------+----------------+-----------------+----------
|           51 |                     1 | ./test/t-main | 0001000000     | 0001000000      | test
|           51 |                     1 | ./test/t-main | 000a000000     | 000a000000      | test
|           51 |                     1 | ./test/t-main | 0064000000     | 0064000000      | test
+--------------+-----------------------+---------------+----------------+-----------------+----------

mysql> SELECT * FROM information_schema.tokudb_lock_waits;

Empty set (0.00 sec)
```

The keys are currently hex dumped.

Now we switch to the other MySQL client with ID 2.

```
mysql> INSERT INTO table VALUES (2),(20),(100);
```

The insert gets blocked since there is a conflict on the primary key with value 100.

The granted TokuDB locks are:

```
mysql> SELECT * FROM information_schema.tokudb_locks;

+--------------+-----------------------+---------------+----------------+-----------------+----------
| locks_trx_id | locks_mysql_thread_id | locks_dname   | locks_key_left | locks_key_right | locks_tab
+--------------+-----------------------+---------------+----------------+-----------------+----------
|           51 |                     1 | ./test/t-main | 0001000000     | 0001000000      | test
|           51 |                     1 | ./test/t-main | 000a000000     | 000a000000      | test
|           51 |                     1 | ./test/t-main | 0064000000     | 0064000000      | test
|           51 |                     1 | ./test/t-main | 0002000000     | 0002000000      | test
|           51 |                     1 | ./test/t-main | 0014000000     | 0014000000      | test
+--------------+-----------------------+---------------+----------------+-----------------+----------
```

The locks that are pending due to a conflict are:

```
SELECT * FROM information_schema.tokudb_lock_waits;

+-------------------+-----------------+-----------------+---------------------+---------------------
| requesting_trx_id | blocking_trx_id | lock_waits_dname | lock_waits_key_left | lock_waits_key_right
+-------------------+-----------------+-----------------+---------------------+---------------------
|                62 |              51 | ./test/t-main    | 0064000000          | 0064000000
+-------------------+-----------------+-----------------+---------------------+---------------------
```

Eventually, the lock for client 2 times out, and we can retrieve a JSON document that describes the conflict.

```
ERROR 1205 (HY000): Lock wait timeout exceeded; try restarting transaction
```

```
mysql> SELECT @@tokudb_last_lock_timeout;

+-----------------------------------------------------------------------------------------------------------------
| @@tokudb_last_lock_timeout
+-----------------------------------------------------------------------------------------------------------------
| "mysql_thread_id":2, "dbname":"./test/t-main", "requesting_txnid":62, "blocking_txnid":51, "key":"0
+-----------------------------------------------------------------------------------------------------------------

rollback;
```

Since transaction 62 was rolled back, all of the locks taken by it are released.

```
mysql> SELECT * FROM information_schema.tokudb_locks;

+--------------+-----------------------+---------------+----------------+-----------------+----------
| locks_trx_id | locks_mysql_thread_id | locks_dname   | locks_key_left | locks_key_right | locks_tab
+--------------+-----------------------+---------------+----------------+-----------------+----------
|           51 |                     1 | ./test/t-main | 0001000000     | 0001000000      | test
|           51 |                     1 | ./test/t-main | 000a000000     | 000a000000      | test
|           51 |                     1 | ./test/t-main | 0064000000     | 0064000000      | test
|           51 |                     2 | ./test/t-main | 0002000000     | 0002000000      | test
|           51 |                     2 | ./test/t-main | 0014000000     | 0014000000      | test
+--------------+-----------------------+---------------+----------------+-----------------+----------
```

6.3 Engine Status

Engine status provides details about the inner workings of TokuDB and can be useful in tuning your particular environment. The engine status can be determined by running the following command:

```
SHOW ENGINE tokudb STATUS;
```

The following is a reference of table status statements:

cachetable: cleaner executions Total number of times the cleaner thread loop has executed.

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

cachetable: 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.

cachetable: evictions Number of blocks evicted from cache.

cachetable: long time waiting on cache pressure Total time, in microseconds, waiting on cache pressure to subside for more than 1 second.

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

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

cachetable: number of long waits on cache pressure The number of times a thread was stalled for more than 1 second due to cache pressure.

cachetable: number of waits on cache pressure The number of times a thread was stalled due to cache pressure.

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: size cachepressure The number of bytes causing cache pressure (the sum of buffers and workdone counters), helps to understand if cleaner threads are keeping up with workload.

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 currently cloned data for checkpoint Amount of memory, in bytes, currently used for cloned nodes. During the checkpoint operation, dirty nodes are cloned prior to serialization/compression, then written to disk. After which, the memory for the cloned block is returned for re-use.

cachetable: size leaf The number of bytes of leaf nodes in the 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 nonleaf The number of bytes of non-leaf 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.

cachetable: time waiting on cache pressure Total time, in microseconds, waiting on cache pressure to subside.

checkpoint: begin time Cumulative time (in microseconds) required to mark all dirty nodes as pending a checkpoint.

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

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

checkpoint: footprint Where the database is in the checkpoint process.

checkpoint: 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.

---

checkpoint: 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.

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

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

checkpoint: long checkpoint begin count The total number of times a checkpoint begin took more than 1 second.

checkpoint: long checkpoint begin time The total time, in microseconds, of long checkpoint begins. A long checkpoint begin is one taking more than 1 second.

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

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

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

checkpoint: time spent during checkpoint (begin and end phases) Time (in seconds) required to complete all checkpoints.

checkpoint: time spent during last checkpoint (begin and end phases) Time (in seconds) required to complete the last checkpoint.

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.

context: promotion blocked by a flush Number of times node rwlock contention was observed within promotion (pinning nodes from root to the buffer to receive the message) because of a buffer flush from parent to child.

context: promotion blocked by a full eviction (should never happen) Number of times node rwlock contention was observed within promotion (pinning nodes from root to the buffer to receive the message) because of a full eviction.

context: promotion blocked by a full fetch (should never happen) Number of times node rwlock contention was observed within promotion (pinning nodes from root to the buffer to receive the message) because of a full fetch.

context: promotion blocked by a message application Number of times node rwlock contention was observed within promotion (pinning nodes from root to the buffer to receive the message) because of message application (applying fresh ancestors messages to a basement node).

context: promotion blocked by a message injection Number of times node rwlock contention was observed within promotion (pinning nodes from root to the buffer to receive the message) because of message injection.

context: promotion blocked by a partial eviction (should never happen) Number of times node rwlock contention was observed within promotion (pinning nodes from root to the buffer to receive the message) because of a partial eviction.

context: promotion blocked by a partial fetch (should never happen) Number of times node rwlock contention was observed within promotion (pinning nodes from root to the buffer to receive the message) because of a partial fetch.

context: promotion blocked by something uninstrumented Number of times node rwlock contention was observed within promotion (pinning nodes from root to the buffer to receive the message) because of something uninstrumented.

context: promotion blocked by the cleaner thread Number of times node rwlock contention was observed within promotion (pinning nodes from root to the buffer to receive the message) because of a cleaner thread.

context: something uninstrumented blocked by something uninstrumented Number of times node rwlock contention was observed for an uninstrumented process because of something uninstrumented.

context: tree traversals blocked by a flush Number of times node rwlock contention was observed while pinning nodes from root to leaf because of a buffer flush from parent to child.

context: tree traversals blocked by a full eviction Number of times node rwlock contention was observed while pinning nodes from root to leaf because of a full eviction.

context: tree traversals blocked by a full fetch Number of times node rwlock contention was observed while pinning nodes from root to leaf because of a full fetch.

context: tree traversals blocked by a message application Number of times node rwlock contention was observed while pinning nodes from root to leaf because of message application (applying fresh ancestors messages to a basement node).

context: tree traversals blocked by a message injection Number of times node rwlock contention was observed while pinning nodes from root to leaf because of message injection.

context: tree traversals blocked by a partial eviction Number of times node rwlock contention was observed while pinning nodes from root to leaf because of a partial eviction.

context: tree traversals blocked by a partial fetch Number of times node rwlock contention was observed while pinning nodes from root to leaf because of a partial fetch.

context: tree traversals blocked by a the cleaner thread Number of times node rwlock contention was observed while pinning nodes from root to leaf because of a cleaner thread.

context: tree traversals blocked by something uninstrumented Number of times node rwlock contention was observed while pinning nodes from root to leaf because of something uninstrumented.

db closes Number of db close operations.

db opens Number of db open operations.

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.

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

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

dictionary multi 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 single recovery log entry for the entire row.

dictionary multi updates fail This is the number of multi-index update 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.

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 ("more than 10 percent of total file system space")
- Less than twice the reserve
- Less than the reserve
- File system is completely full

filesystem: ENOSPC redzone state The state of how much disk space exists with respect to the red zone value. Valid values are:

0 Space is available

1 Warning, with 2x of redzone value. Operations are allowed, but engine status prints a warning.

2 In red zone, insert operations are blocked

3 All operations are blocked

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

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

filesystem: long fsync count This is the total number of times the database has flushed the operating system's file buffers to disk and this operation required more than 1 second.

filesystem: long fsync time This the total time, in microseconds, used to fsync to disk when the operation required more than 1 second.

filesystem: 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.

filesystem: number of operations rejected by enospc prevention (red zone) This is the number of database inserts that have been rejected because the amount of disk free space was less than the reserve.

filesystem: number of write operations that returned ENOSPC 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.

filesystem: 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.

ft: basements decompressed as a target of a query Number of basement nodes decompressed for queries.

ft: basements decompressed for prefetch Number of basement nodes decompressed by a prefetch thread.

ft: basements decompressed for prelocked range Number of basement nodes decompressed by queries aggressively.

ft: basements decompressed for write Number of basement nodes decompressed for writes.

ft: basement nodes deserialized with fixed-keysize The number of basement nodes deserialized where all keys had the same size, leaving the basement in a format that is optimal for in-memory workloads.

ft: basement nodes deserialized with variable-keysize The number of basement nodes deserialized where all keys did not have the same size, and thus ineligible for an in-memory optimization.

ft: basements fetched as a target of a query (bytes) Number of basement node bytes fetched from disk for queries.

ft: basements fetched as a target of a query Number of basement nodes fetched from disk for queries.

ft: basements fetched as a target of a query (seconds) Number of seconds waiting for IO when fetching basement nodes from disk for queries.

ft: basements fetched for prefetch (bytes) Number of basement node bytes fetched from disk by a prefetch thread.

ft: basements fetched for prefetch Number of basement nodes fetched from disk by a prefetch thread.

ft: basements fetched for prefetch (seconds) Number of seconds waiting for IO when fetching basement nodes from disk by a prefetch thread.

ft: basements fetched for prelocked range (bytes) Number of basement node bytes fetched from disk aggressively.

ft: basements fetched for prelocked range Number of basement nodes fetched from disk aggressively.

ft: basements fetched for prelocked range (seconds) Number of seconds waiting for IO when fetching basement nodes from disk aggressively.

ft: basements fetched for write (bytes) Number of basement node bytes fetched from disk for writes.

ft: basements fetched for write Number of basement nodes fetched from disk for writes.

ft: basements fetched for write (seconds) Number of seconds waiting for IO when fetching basement nodes from disk for writes.

ft: broadcast messages injected at root How many broadcast messages injected at root.

ft: buffers decompressed as a target of a query Number of buffers decompressed for queries.

ft: buffers decompressed for prefetch Number of buffers decompressed by a prefetch thread.

ft: buffers decompressed for prelocked range Number of buffers decompressed by queries aggressively.

ft: buffers decompressed for write Number of buffers decompressed for writes.

ft: buffers fetched as a target of a query (bytes) Number of buffer bytes fetched from disk for queries.

ft: buffers fetched as a target of a query Number of buffers fetched from disk for queries.

ft: buffers fetched as a target of a query (seconds) Number of seconds waiting for IO when fetching buffers from disk for queries.

ft: buffers fetched for prefetch (bytes) Number of buffer bytes fetched from disk by a prefetch thread.

ft: buffers fetched for prefetch Number of buffers fetched from disk by a prefetch thread.

ft: buffers fetched for prefetch (seconds) Number of seconds waiting for IO when fetching buffers from disk by a prefetch thread.

ft: buffers fetched for prelocked range (bytes) Number of buffer bytes fetched from disk aggressively.

ft: buffers fetched for prelocked range Number of buffers fetched from disk aggressively.

ft: buffers fetched for prelocked range (seconds) Number of seconds waiting for IO when fetching buffers from disk aggressively.

ft: buffers fetched for write (bytes) Number of buffer bytes fetched from disk for writes.

ft: buffers fetched for write Number of buffers fetched from disk for writes.

ft: buffers fetched for write (seconds) Number of seconds waiting for IO when fetching buffers from disk for writes.

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

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

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

ft: 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).

ft: leaf compression to memory (seconds) Total time, in seconds, spent compressing leaf nodes.

ft: leaf decompression to memory (seconds) Total time, in seconds, spent decompressing leaf nodes.

ft: leaf deserialization to memory (seconds) Total time, in seconds, spent deserializing leaf nodes.

ft: leaf node full evictions (bytes) The number of bytes freed by evicting full leaf nodes from the cache.

ft: leaf node full evictions The number of times a full leaf node was evicted from the cache.

ft: leaf node partial evictions (bytes) The number of bytes freed by evicting partitions of leaf nodes from the cache.

ft: leaf node partial evictions The number of times a partition of a leaf node was evicted from the cache.

ft: leaf nodes created Number of leaf nodes created.

ft: leaf nodes destroyed Number of leaf nodes destroyed.

ft: leaf nodes flushed to disk (for checkpoint) (bytes) Number of bytes of leaf nodes flushed to disk for checkpoint.

ft: leaf nodes flushed to disk (for checkpoint) Number of leaf nodes flushed to disk for checkpoint.

ft: leaf nodes flushed to disk (for checkpoint) (seconds) Number of seconds waiting for IO when writing leaf nodes flushed to disk for checkpoint.

ft: leaf nodes flushed to disk (for checkpoint) (uncompressed bytes) Number of uncompressed bytes of leaf nodes flushed to disk for checkpoint.

ft: leaf nodes flushed to disk (not for checkpoint) (bytes) Number of bytes of leaf nodes flushed to disk, not for checkpoint.

ft: leaf nodes flushed to disk (not for checkpoint) Number of leaf nodes flushed to disk, not for checkpoint.

ft: leaf nodes flushed to disk (not for checkpoint) (seconds) Number of seconds waiting for IO when writing leaf nodes flushed to disk, not for checkpoint.

ft: leaf nodes flushed to disk (not for checkpoint) (uncompressed bytes) Number of bytes of leaf nodes flushed to disk, not for checkpoint.

ft: leaf serialization to memory (seconds) Total time, in seconds, spent serializing leaf nodes.

ft: messages ignored by leaf due to msn The number of messages that were ignored by a leaf because it had already been applied.

ft: messages injected at root How many messages injected at root.

ft: nonleaf compression to memory (seconds) Total time, in seconds, spent compressing non leaf nodes.

ft: nonleaf decompression to memory (seconds) Total time, in seconds, spent decompressing non leaf nodes.

ft: nonleaf deserialization to memory (seconds) Total time, in seconds, spent deserializing non leaf nodes.

ft: nonleaf node full evictions (bytes) The number of bytes freed by evicting full nonleaf nodes from the cache.

ft: nonleaf node full evictions The number of times a full nonleaf node was evicted from the cache.

ft: nonleaf node partial evictions (bytes) The number of bytes freed by evicting partitions of nonleaf nodes from the cache.

ft: nonleaf node partial evictions The number of times a partition of a nonleaf node was evicted from the cache.

ft: nonleaf nodes created Number of nonleaf nodes created.

ft: nonleaf nodes destroyed Number of nonleaf nodes destroyed.

ft: nonleaf nodes flushed to disk (for checkpoint) (bytes) Number of bytes of nonleaf nodes flushed to disk for checkpoint.

ft: nonleaf nodes flushed to disk (for checkpoint) Number of nonleaf nodes flushed to disk for checkpoint.

ft: nonleaf nodes flushed to disk (for checkpoint) (seconds) Number of seconds waiting for IO when writing non-leaf nodes flushed to disk for checkpoint.

ft: nonleaf nodes flushed to disk (for checkpoint) (uncompressed bytes) Number of uncompressed bytes of non-leaf nodes flushed to disk for checkpoint.

ft: nonleaf nodes flushed to disk (not for checkpoint) (bytes) Number of bytes of nonleaf nodes flushed to disk, not for checkpoint.

ft: nonleaf nodes flushed to disk (not for checkpoint) Number of nonleaf nodes flushed to disk, not for checkpoint.

ft: nonleaf nodes flushed to disk (not for checkpoint) (seconds) Number of seconds waiting for IO when writing nonleaf nodes flushed to disk, not for check- point.

ft: nonleaf nodes flushed to disk (not for checkpoint) (uncompressed bytes) Number of uncompressed bytes of nonleaf nodes flushed to disk, not for checkpoint.

ft: nonleaf serialization to memory (seconds) Total time, in seconds, spent serializing non leaf nodes.

ft: pivots fetched for prefetch (bytes) Number of bytes of pivot nodes fetched by a prefetch thread.

ft: pivots fetched for prefetch Number of pivot nodes fetched by a prefetch thread.

ft: pivots fetched for prefetch (seconds) Number seconds waiting for IO when fetching pivot nodes by a prefetch thread.

ft: pivots fetched for query (bytes) Number of bytes of pivot nodes fetched for queries.

ft: pivots fetched for query Number of pivot nodes fetched for queries.

ft: pivots fetched for query (seconds) Number of seconds waiting for IO when fetching pivot nodes for queries.

ft: pivots fetched for write (bytes) Number of bytes of pivot nodes fetched for writes.

ft: pivots fetched for write Number of pivot nodes fetched for writes.

ft: pivots fetched for write (seconds) Number of seconds waiting for IO when fetching pivot nodes for writes.

ft: promotion: h1 roots injected into Number of times a message stopped at a root with height 1.

ft: promotion: injections at depth 0 Number of times a message stopped at depth 0.

ft: promotion: injections at depth 1 Number of times a message stopped at depth 1.

ft: promotion: injections at depth 2 Number of times a message stopped at depth 2.

ft: promotion: injections at depth 3 Number of times a message stopped at depth 3.

ft: promotion: injections lower than depth 3 Number of times a message was promoted past depth 3.

ft: promotion: leaf roots injected into Number of times a message stopped at a root with height 0.

ft: promotion: roots split Number of times the root split during promotion.

ft: promotion: stopped anyway, after locking the child Number of times a message stopped before a child which had been locked.

ft: promotion: stopped at height 1 Number of times a message stopped because it had reached height 1.

ft: promotion: stopped because of a nonempty buffer Number of times a message stopped because it reached a nonempty buffer.

ft: promotion: stopped because the child was locked or not at all in memory Number of times a message stopped because it could not cheaply get access to a child.

ft: promotion: stopped because the child was not fully in memory Number of times a message stopped because it could not cheaply get access to a child.

ft: promotion: succeeded in using the rightmost leaf shortcut Rightmost insertions used the rightmost-leaf pin path, meaning that the Fractal Tree index detected and properly optimized rightmost inserts.

ft: promotion: tried the rightmost leaf shorcut but failed (child reactive) Rightmost insertions did not use the rightmost-leaf pin path, due to the leaf being too large (needed to split).

ft: promotion: tried the rightmost leaf shorcut but failed (out-of-bounds) Rightmost insertions did not use the rightmost-leaf pin path, due to the insert not actually being into the rightmost leaf node.

ft: searches requiring more tries than the height of the tree Number of searches that required more tries than the height of the tree.

ft: searches requiring more tries than the height of the tree plus three Number of searches that required more tries than the height of the tree plus three.

ft: total search retries due to TRY AGAIN Total number of search retries due to TRY AGAIN.

ft: uncompressed / compressed bytes written (leaf) Ratio of uncompressed bytes (in-memory) to compressed bytes (on-disk) for leaf nodes.

ft: uncompressed / compressed bytes written (nonleaf) Ratio of uncompressed bytes (in-memory) to compressed bytes (on-disk) for nonleaf nodes.

ft: uncompressed / compressed bytes written (overall) Ratio of uncompressed bytes (in-memory) to compressed bytes (on-disk) for all nodes.

ft flusher: cleaner thread leaf merges in progress The number of cleaner thread leaf merges in progress.

ft flusher: cleaner thread leaf merges successful The number of times the cleaner thread successfully merges a leaf.

ft flusher: height-greater-than-one nodes flushed by cleaner thread Number of nodes of height > 1 whose message buffers are flushed by cleaner thread.

ft flusher: height-one nodes flushed by cleaner thread Number of nodes of height one whose message buffers are flushed by cleaner thread.

ft flusher: leaf node balances Number of times a leaf node is balanced.

ft flusher: leaf node merges Number of times leaf nodes are merged.

ft flusher: leaf node splits Number of leaf nodes split.

ft flusher: max bytes in a buffer flushed by cleaner thread Max number of bytes in message buffer flushed by cleaner thread.

ft flusher: max workdone in a buffer flushed by cleaner thread Max workdone value of any message buffer flushed by cleaner thread.

ft flusher: min bytes in a buffer flushed by cleaner thread Min number of bytes in message buffer flushed by cleaner thread.

ft flusher: min workdone in a buffer flushed by cleaner thread Min workdone value of any message buffer flushed by cleaner thread.

ft flusher: nodes cleaned which had empty buffers Number of nodes that are selected by cleaner, but whose buffers are empty.

ft flusher: nodes dirtied by cleaner thread Number of nodes that are made dirty by the cleaner thread.

ft flusher: nodes dirtied by cleaner thread leaf merges The number of nodes dirtied by the "flush from root" process to merge a leaf node.

ft flusher: nonleaf node merges Number of times nonleaf nodes are merged.

ft flusher: nonleaf node splits Number of nonleaf nodes split.

ft flusher: number of flushes that read something off disk Number of flushes that had to read a child (or part) off disk.

ft flusher: number of flushes that triggered 1 cascading flush Number of flushes that triggered 1 cascading flush.

ft flusher: number of flushes that triggered 2 cascading flushes Number of flushes that triggered 2 cascading flushes.

ft flusher: number of flushes that triggered 3 cascading flushes Number of flushes that triggered 3 cascading flushes.

ft flusher: number of flushes that triggered 4 cascading flushes Number of flushes that triggered 4 cascading flushes.

ft flusher: number of flushes that triggered 5 cascading flushes Number of flushes that triggered 5 cascading flushes.

ft flusher: number of flushes that triggered another flush in child Number of flushes that triggered another flush in the child.

ft flusher: number of flushes that triggered over 5 cascading flushes Number of flushes that triggered more than 5 cascading flushes.

ft flusher: number of in memory flushes Number of in-memory flushes.

ft flusher: times cleaner thread tries to merge a leaf The number of times the cleaner thread tries to merge a leaf.

ft flusher: total bytes in buffers flushed by cleaner thread Total number of bytes in message buffers flushed by cleaner thread.

ft flusher: total nodes potentially flushed by cleaner thread Total number of nodes whose buffers are potentially flushed by cleaner thread.

ft flusher: total number of flushes done by flusher threads or cleaner threads Total number of flushes done by flusher threads or cleaner threads.

ft flusher: total workdone in buffers flushed by cleaner thread Total workdone value of message buffers flushed by cleaner thread.

handlerton: primary key bytes inserted Total number of bytes inserted into all primary key indexes.

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

hot: operations aborted The number of HOT operations that have been aborted.

hot: operations ever started The number of HOT operations that have begun.

hot: operations successfully completed The number of HOT operations that have successfully completed.

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

indexer: number of calls to indexer->abort() This is the number of indexers that were aborted.

indexer: number of calls to indexer->build() failed This is the total number of times that indexes were unable to be created using a indexer

indexer: number of calls to indexer->build() succeeded This is the total number of times that indexes were created using a indexer.

indexer: number of calls to indexer->close() that failed This is the number of indexers that were unable to create the requested index(es).

indexer: number of calls to indexer->close() that succeeded This is the number of indexers that successfully created the requested index(es).

indexer: number of calls to toku indexer create indexer() that failed This is the number of times a indexer was requested but could not be created.

indexer: number of indexers currently in existence This is the number of indexers that currently exist.

indexer: number of indexers successfully created This is the number of times one of our internal objects, a indexer, has been created.

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

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

le: size of leafentries after garbage collection (during message application) Total number of bytes of leaf nodes data after performing garbage collection for non-flush events.

le: size of leafentries after garbage collection (outside message application) Total number of bytes of leaf nodes data after performing garbage collection for flush events.

le: size of leafentries before garbage collection (during message application) Total number of bytes of leaf nodes data before performing garbage collection for non-flush events.

le: size of leafentries before garbage collection (outside message application) Total number of bytes of leaf nodes data before performing garbage collection for flush events.

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

loader: number of calls to loader->abort() This is the number of loaders that were aborted.

loader: number of calls to loader->close() that failed This is the number of loaders that were unable to create the requested table.

loader: number of calls to loader->close() that succeeded This is the number of loaders that successfully created the requested table.

loader: number of calls to loader->put() failed This is the total number of rows that were unable to be inserted using a loader.

loader: number of calls to loader->put() succeeded This is the total number of rows that were inserted using a loader.

loader: number of calls to toku loader create loader() that failed This is the number of times a loader was requested but could not be created.

loader: number of loaders currently in existence This is the number of loaders that currently exist.

loader: number of loaders successfully created This is the number of times one of our internal objects, a loader, has been created.

locktree: latest post-escalation memory size Size of the locktree, in bytes, after most current locktree escalation.

locktree: long time waiting for locks Total time, in microseconds, of the long waits.

locktree: long time waiting on lock escalation Total time, in microseconds, of the long waits for lock escalation to free up memory.

locktree: memory size Count, in bytes, that the locktree is currently using.

locktree: memory size limit Maximum number of bytes that the locktree is allowed to use.

locktree: number of lock timeouts Count of the number of times that a lock request timed out.

locktree: number of locktrees eligible for the STO Number of locktrees eligible for "single transaction optimizations".

locktree: number of locktrees open now Number of locktrees currently open.

locktree: number of lock waits Number of times that a lock request could not be acquired because of a conflict with some other transaction.

locktree: number of long lock waits Number of lock waits greater than 1 second in duration.

locktree: number of long waits on lock escalation Number of times that a client thread had to wait on lock escalation and the wait time was greater than 1 second.

locktree: number of pending lock requests Number of requesters waiting for a lock grant.

locktree: number of times a locktree ended the STO early Total number of times a "single transaction optimization" was ended early due to another trans- action starting.

locktree: number of times lock escalation ran Number of times the locktree needed to run lock escalation to reduce its memory footprint.

locktree: number of waits on lock escalation When the sum of the sizes of locks taken reaches the lock tree limit, we run lock escalation on a background thread. The clients threads need to wait for escalation to consolidate locks and free up memory. This counter counts the number of times a client thread has to wait on lock escalation.

locktree: time spent ending the STO early (seconds) Total number of seconds ending "single transaction optimizations".

locktree: time spent running escalation (seconds) Total number of seconds spent performing locktree escalation.

locktree: time waiting for locks Total time, in microseconds, spend by some client waiting for a lock conflict to be resolved.

locktree: time waiting on lock escalation Total time, in microseconds, that a client thread spent waiting for lock escalation to free up memory.

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

logger: number of long logger write operations Number of times a logger write operation required 100ms or more.

logger: writes (bytes) Number of bytes the logger has written to disk.

logger: writes Number of times the logger has written to disk.

logger: writes (seconds) Number of seconds waiting for IO when writing logs to disk.

logger: writes (uncompressed bytes) Number of uncompressed the logger has written to disk.

max open dbs Max number of simultaneously open DBs.

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

memory: largest attempted allocation size Largest number of bytes in a single successful malloc() operation.

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

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

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

memory: number of bytes requested Total number of bytes requested from mallocator.

memory: number of bytes used (requested + overhead) Total number of bytes allocated by mallocator.

memory: number of free operations Number of calls to free().

memory: number of malloc operations Number of calls to malloc().

memory: number of malloc operations that failed Number of failed calls to malloc().

memory: number of realloc operations Number of calls to realloc().

memory: number of realloc operations that failed Number of failed calls to realloc().

memory: size of the last failed allocation attempt Largest number of bytes in a single failed malloc() operation.

num open dbs now Number of currently open DBs.

period, in ms, that recovery log is automatically fsynced fsync() frequency in milliseconds.

time now Current date/time on server.

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

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.

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

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

txn: begin read only Number of read only transactions started.

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

6.4 Global Status

The `information_schema.global_status` table provides details about the inner workings of TokuDB and can be useful in tuning your particular environment. The statuses can be determined with the following command:

```
SELECT * FROM information_schema.global_status;
```

EThe following global status parameters are available:

TOKUDB_BASEMENTS_DECOMPRESSED_FOR_WRITE Number of basement nodes decompressed for writes.

TOKUDB_BASEMENTS_DECOMPRESSED_PREFETCH Number of basement nodes decompressed by a prefetch thread.

TOKUDB_BASEMENTS_DECOMPRESSED_PRELOCKED_RANGE Number of basement nodes decompressed by queries aggressively.

TOKUDB_BASEMENTS_DECOMPRESSED_TARGET_QUERY Number of basement nodes decompressed for queries.

TOKUDB_BASEMENT_DESERIALIZATION_FIXED_KEY Number of basement nodes deserialized where all keys had the same size, leaving the basement in a format that is optimal for in-memory workloads.

TOKUDB_BASEMENT_DESERIALIZATION_VARIABLE_KEY Number of basement nodes deserialized where all keys did not have the same size, and thus ineligible for an in-memory optimization.

TOKUDB_BASEMENTS_FETCHED_FOR WRITE_BYTES Number of basement node bytes fetched from disk for writes.

TOKUDB_BASEMENTS_FETCHED_FOR WRITE Number of basement nodes fetched from disk for writes.

TOKUDB_BASEMENTS_FETCHED_FOR WRITE_SECONDS Number of seconds waiting for IO when fetching basement nodes from disk for writes.

TOKUDB_BASEMENTS_FETCHED_PREFETCH_BYTES Number of basement node bytes fetched from disk by a prefetch thread.

TOKUDB_BASEMENTS_FETCHED_PREFETCH Number of basement nodes fetched from disk by a prefetch thread.

TOKUDB_BASEMENTS_FETCHED_PREFETCH_SECONDS Number of seconds waiting for IO when fetching basement nodes from disk by a prefetch thread.

TOKUDB_BASEMENTS_FETCHED_PRELOCKED_RANGE_BYTES Number of basement node bytes fetched from disk aggressively.

TOKUDB_BASEMENTS_FETCHED_PRELOCKED_RANGE Number of basement nodes fetched from disk aggressively.

TOKUDB_BASEMENTS_FETCHED_PRELOCKED_RANGE_SECONDS Number of seconds waiting for IO when fetching basement nodes from disk aggressively.

TOKUDB_BASEMENTS_FETCHED_TARGET_QUERY_BYTES Number of basement node bytes fetched from disk for queries.

TOKUDB_BASEMENTS_FETCHED_TARGET_QUERY Number of basement nodes fetched from disk for queries.

TOKUDB_BASEMENTS_FETCHED_TARGET_QUERY_SECONDS Number of seconds waiting for IO when fetching basement nodes from disk for queries.

TOKUDB_BROADCAST_MESSAGES_INJECTED_AT_ROOT How many broadcast messages injected at root.

TOKUDB_BUFFERS_DECOMPRESSED_FOR_WRITE Number of buffers decompressed for writes.

TOKUDB_BUFFERS_DECOMPRESSED_PREFETCH Number of buffers decompressed by a prefetch thread.

TOKUDB_BUFFERS_DECOMPRESSED_PRELOCKED_RANGE Number of buffers decompressed by queries aggressively.

TOKUDB_BUFFERS_DECOMPRESSED_TARGET_QUERY Number of buffers decompressed for queries.

TOKUDB_BUFFERS_FETCHED_FOR_WRITE_BYTES Number of buffer bytes fetched from disk for writes.

TOKUDB_BUFFERS_FETCHED_FOR_WRITE Number of buffers fetched from disk for writes.

TOKUDB_BUFFERS_FETCHED_FOR_WRITE_SECONDS Number of seconds waiting for IO when fetching buffers from disk for writes.

TOKUDB_BUFFERS_FETCHED_PREFETCH_BYTES Number of buffer bytes fetched from disk by a prefetch thread.

TOKUDB_BUFFERS_FETCHED_PREFETCH Number of buffers fetched from disk by a prefetch thread.

TOKUDB_BUFFERS_FETCHED_PREFETCH_SECONDS Number of seconds waiting for IO when fetching buffers from disk by a prefetch thread.

TOKUDB_BUFFERS_FETCHED_PRELOCKED_RANGE_BYTES Number of buffer bytes fetched from disk aggressively.

TOKUDB_BUFFERS_FETCHED_PRELOCKED_RANGE Number of buffers fetched from disk aggressively.

TOKUDB_BUFFERS_FETCHED_PRELOCKED_RANGE_SECONDS Number of seconds waiting for IO when fetching buffers from disk aggressively.

TOKUDB_BUFFERS_FETCHED_TARGET_QUERY_BYTES Number of buffer bytes fetched from disk for queries.

TOKUDB_BUFFERS_FETCHED_TARGET_QUERY Number of buffers fetched from disk for queries.

TOKUDB_BUFFERS_FETCHED_TARGET_QUERY_SECONDS Number of seconds waiting for IO when fetching buffers from disk for queries.

TOKUDB_CACHETABLE_CLEANER_EXECUTIONS Total number of times the cleaner thread loop has executed.

TOKUDB_CACHETABLE_CLEANER_ITERATIONS This is the number of cleaner operations that are performed every cleaner period.

TOKUDB_CACHETABLE_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.

TOKUDB_CACHETABLE_EVICTIONS Number of blocks evicted from cache.

TOKUDB_CACHETABLE_LONG_WAIT_PRESSURE_COUNT The number of times a thread was stalled for more than 1 second due to cache pressure.

TOKUDB_CACHETABLE_LONG_WAIT_PRESSURE_TIME Total time, in microseconds, waiting on cache pressure to subside for more than 1 second.

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

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

TOKUDB_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.

TOKUDB_CACHETABLE_SIZE_CACHEPRESSURE The number of bytes causing cache pressure (the sum of buffers and workdone counters), helps to understand if cleaner threads are keeping up with workload.

TOKUDB_CACHETABLE_SIZE_CLONED Amount of memory, in bytes, currently used for cloned nodes. During the checkpoint operation, dirty nodes are cloned prior to serialization/compression, then written to disk. After which, the memory for the cloned block is returned for re-use.

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

TOKUDB_CACHETABLE_SIZE_LEAF The number of bytes of leaf nodes in the cache.

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

TOKUDB_CACHETABLE_SIZE_NONLEAF The number of bytes of nonleaf nodes in the cache.

TOKUDB_CACHETABLE_SIZE_ROLLBACK The number of bytes of rollback nodes in the cache.

TOKUDB_CACHETABLE_SIZE_WRITING This is the number of bytes that are currently queued up to be written to disk.

TOKUDB_CACHETABLE_WAIT_PRESSURE_COUNT The number of times a thread was stalled due to cache pressure.

TOKUDB_CACHETABLE_WAIT_PRESSURE TIME Total time, in microseconds, waiting on cache pressure to subside.

TOKUDB_CHECKPOINT_BEGIN_TIME Cumulative time (in microseconds) required to mark all dirty nodes as pending a checkpoint.

TOKUDB_CHECKPOINT_DURATION_LAST Time (in seconds) required to complete the last checkpoint.

TOKUDB_CHECKPOINT_DURATION Time (in seconds) required to complete all checkpoints.

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

TOKUDB_CHECKPOINT_LAST_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.)

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

TOKUDB_CHECKPOINT_LAST_COMPLETE_ENDED This is the time the last complete checkpoint ended.

TOKUDB_CHECKPOINT_LONG_CHECKPOINT_BEGIN_COUNT The total number of times a checkpoint begin took more than 1 second.

TOKUDB_CHECKPOINT_LONG_CHECKPOINT_BEGIN_TIME The total time, in microseconds, of long checkpoint begins. A long checkpoint begin is one taking more than 1 second.

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

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

TOKUDB_DB_CLOSES Number of db close operations.

TOKUDB_DB_OPEN_CURRENT Number of currently open DBs.

TOKUDB_DB_OPEN_MAX Max number of simultaneously open DBs.

TOKUDB_DB_OPENS Number of db open operations.

TOKUDB_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).

TOKUDB_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.

TOKUDB_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.

TOKUDB_FILESYSTEM_FSYNC_NUM This is the total number of times the database has flushed the operating system's file buffers to disk.

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

TOKUDB_FILESYSTEM_LONG_FSYNC_NUM This is the total number of times the database has flushed the operating system's file buffers to disk and this operation required more than 1 second.

TOKUDB_FILESYSTEM_LONG_FSYNC_TIME This the total time, in microseconds, used to fsync to disk when the operation required more than 1 second.

TOKUDB_FILESYSTEM_THREADS_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.

TOKUDB_LEAF_COMPRESSION_TO_MEMORY_SECONDS Total time, in seconds, spent compressing leaf nodes.

TOKUDB_LEAF_DECOMPRESSION_TO_MEMORY_SECONDS Total time, in seconds, spent decompressing leaf nodes.

TOKUDB_LEAF_DESERIALIZATION_TO_MEMORY_SECONDS Total time, in seconds, spent deserializing leaf nodes.

TOKUDB_LEAF_NODE_COMPRESSION_RATIO Ratio of uncompressed bytes (in-memory) to compressed bytes (on-disk) for leaf nodes.

TOKUDB_LEAF_NODE_FULL_EVICTIONS_BYTES The number of bytes freed by evicting full leaf nodes from the cache.

TOKUDB_LEAF_NODE_FULL_EVICTIONS The number of times a full leaf node was evicted from the cache.

TOKUDB_LEAF_NODE_PARTIAL_EVICTIONS_BYTES The number of bytes freed by evicting partitions of leaf nodes from the cache.

TOKUDB_LEAF_NODE_PARTIAL_EVICTIONS The number of times a partition of a leaf node was evicted from the cache.

TOKUDB_LEAF_NODES_CREATED Number of leaf nodes created.

TOKUDB_LEAF_NODES_DESTROYED Number of leaf nodes destroyed.

TOKUDB_LEAF_NODES_FLUSHED_CHECKPOINT_BYTES Number of bytes of leaf nodes flushed to disk for checkpoint.

TOKUDB_LEAF_NODES_FLUSHED_CHECKPOINT Number of leaf nodes flushed to disk for checkpoint.

TOKUDB_LEAF_NODES_FLUSHED_CHECKPOINT_SECONDS Number of seconds waiting for IO when writing leaf nodes flushed to disk for checkpoint.

TOKUDB_LEAF_NODES_FLUSHED_CHECKPOINT_UNCOMPRESSED BYTES Number of uncompressed bytes of leaf nodes flushed to disk for checkpoint.

TOKUDB_LEAF_NODES_FLUSHED_NOT_CHECKPOINT_BYTES Number of bytes of leaf nodes flushed to disk, not for checkpoint.

TOKUDB_LEAF_NODES_FLUSHED_NOT_CHECKPOINT Number of leaf nodes flushed to disk, not for checkpoint.

TOKUDB_LEAF_NODES_FLUSHED_NOT_CHECKPOINT_SECONDS Number of seconds waiting for IO when writing leaf nodes flushed to disk, not for checkpoint.

TOKUDB_LEAF_NODES_FLUSHED_NOT_CHECKPOINT_UNCOMPRESSED_BYTES Number of bytes of leaf nodes flushed to disk, not for checkpoint.

TOKUDB_LEAF_SERIALIZATION_TO_MEMORY_SECONDS Total time, in seconds, spent serializing leaf nodes.

TOKUDB_LOADER_NUM_CREATED This is the number of times one of our internal objects, a loader, has been created.

TOKUDB_LOADER_NUM_CURRENT This is the number of loaders that currently exist.

TOKUDB_LOADER_NUM_MAX This is the maximum number of loaders that ever existed simultaneously.

TOKUDB_LOCKTREE_ESCALATION_NUM Number of times the locktree needed to run lock escalation to reduce its memory footprint.

TOKUDB_LOCKTREE_ESCALATION_SECONDS Total number of seconds spent performing locktree escalation.

TOKUDB_LOCKTREE_LATEST_POST_ESCALATION_MEMORY_SIZE Size of the locktree, in bytes, after most current locktree escalation.

TOKUDB_LOCKTREE_LONG_WAIT_COUNT Number of lock waits greater than 1 second in duration.

TOKUDB_LOCKTREE_LONG_WAIT_ESCALATION_COUNT Number of times that a client thread had to wait on lock escalation and the wait time was greater than 1 second.

TOKUDB_LOCKTREE_LONG_WAIT_ESCALATION_TIME Total time, in microseconds, of the long waits for lock escalation to free up memory.

TOKUDB_LOCKTREE_LONG_WAIT_TIME Total time, in microseconds, of the long waits.

TOKUDB_LOCKTREE_MEMORY_SIZE Count, in bytes, that the locktree is currently using.

TOKUDB_LOCKTREE_MEMORY_SIZE_LIMIT Maximum number of bytes that the locktree is allowed to use.

TOKUDB_LOCKTREE_OPEN_CURRENT Number of locktrees currently open.

TOKUDB_LOCKTREE_PENDING_LOCK_REQUESTS Number of requesters waiting for a lock grant.

TOKUDB_LOCKTREE_STO_ELIGIBLE_NUM Number of locktrees eligible for "single transaction optimizations".

TOKUDB_LOCKTREE_STO_ENDED_NUM Total number of times a "single transaction optimization" was ended early due to another transaction starting.

TOKUDB_LOCKTREE_STO_ENDED_SECONDS Total number of seconds ending "single transaction optimizations".

TOKUDB_LOCKTREE_TIMEOUT_COUNT Count of the number of times that a lock request timed out.

TOKUDB_LOCKTREE_WAIT_COUNT Number of times that a lock request could not be acquired because of a conflict with some other transaction.

TOKUDB_LOCKTREE_WAIT_ESCALATION_COUNT When the sum of the sizes of locks taken reaches the lock tree limit, we run lock escalation on a background thread. The clients threads need to wait for escalation to consolidate locks and free up memory. This counter counts the number of times a client thread has to wait on lock escalation.

TOKUDB_LOCKTREE_WAIT_ESCALATION_TIME Total time, in microseconds, that a client thread spent waiting for lock escalation to free up memory.

TOKUDB_LOCKTREE_WAIT_TIME Total time, in microseconds, spend by some client waiting for a lock conflict to be resolved.

TOKUDB_LOGGER_WAIT_LONG Number of times a logger write operation required 100ms or more.

TOKUDB_LOGGER_WRITES_BYTES Number of bytes the logger has written to disk.

TOKUDB_LOGGER_WRITES Number of times the logger has written to disk.

TOKUDB_LOGGER_WRITES_SECONDS Number of seconds waiting for IO when writing logs to disk.

TOKUDB_LOGGER_WRITES_UNCOMPRESSED_BYTES Number of uncompressed the logger has written to disk.

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

TOKUDB_MESSAGES_FLUSHED_FROM_H1_TO_LEAVES_BYTES How many bytes of messages flushed from h1 nodes to leaves.

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

TOKUDB_MESSAGES_INJECTED_AT_ROOT_BYTES How many bytes of messages injected at root (for all trees).

TOKUDB_MESSAGES_INJECTED_AT_ROOT How many messages injected at root.

TOKUDB_MESSAGES_IN_TREES_ESTIMATE_BYTES How many bytes of messages currently in trees (estimate).

TOKUDB_NONLEAF_COMPRESSION_TO_MEMORY_SECONDS Total time, in seconds, spent compressing non leaf nodes.

TOKUDB_NONLEAF_DECOMPRESSION_TO_MEMORY_SECONDS Total time, in seconds, spent decompressing non leaf nodes.

TOKUDB_NONLEAF_DESERIALIZATION_TO_MEMORY_SECONDS Total time, in seconds, spent deserializing non leaf nodes.

TOKUDB_NONLEAF_NODE_COMPRESSION_RATIO Ratio of uncompressed bytes (in-memory) to compressed bytes (on-disk) for nonleaf nodes.

TOKUDB_NONLEAF_NODE_FULL_EVICTIONS_BYTES The number of bytes freed by evicting full nonleaf nodes from the cache.

TOKUDB_NONLEAF_NODE_FULL_EVICTIONS The number of times a full nonleaf node was evicted from the cache.

TOKUDB_NONLEAF_NODE_PARTIAL_EVICTIONS_BYTES The number of bytes freed by evicting partitions of nonleaf nodes from the cache.

TOKUDB_NONLEAF_NODE_PARTIAL_EVICTIONS The number of times a partition of a nonleaf node was evicted from the cache.

TOKUDB_NONLEAF_NODES_CREATED Number of nonleaf nodes created.

TOKUDB_NONLEAF_NODES_DESTROYED Number of nonleaf nodes destroyed.

TOKUDB_NONLEAF_NODES_FLUSHED_TO_DISK_CHECKPOINT_BYTES Number of bytes of nonleaf nodes flushed to disk for checkpoint.

TOKUDB_NONLEAF_NODES_FLUSHED_TO_DISK_CHECKPOINT Number of nonleaf nodes flushed to disk for checkpoint.

TOKUDB_NONLEAF_NODES_FLUSHED_TO_DISK_CHECKPOINT_SECONDS Number of seconds waiting for IO when writing nonleaf nodes flushed to disk for checkpoint.

TOKUDB_NONLEAF_NODES_FLUSHED_TO_DISK_CHECKPOINT_UNCOMPRESSED_BYTES Number of uncompressed bytes of nonleaf nodes flushed to disk for checkpoint.

TOKUDB_NONLEAF_NODES_FLUSHED_TO_DISK_NOT_CHECKPOINT_BYTES Number of bytes of nonleaf nodes flushed to disk, not for checkpoint.

TOKUDB_NONLEAF_NODES_FLUSHED_TO_DISK_NOT_CHECKPOINT Number of nonleaf nodes flushed to disk, not for checkpoint.

TOKUDB_NONLEAF_NODES_FLUSHED_TO_DISK_NOT_CHECKPOINT_SECONDS Number of seconds waiting for IO when writing nonleaf nodes flushed to disk, not for check- point.

TOKUDB_NONLEAF_NODES_FLUSHED_TO_DISK_NOT_CHECKPOINT_UNCOMPRESSED_BYTES Number of uncompressed bytes of nonleaf nodes flushed to disk, not for checkpoint.

TOKUDB_NONLEAF_SERIALIZATION_TO_MEMORY_SECONDS Total time, in seconds, spent serializing non leaf nodes.

TOKUDB_OVERALL_NODE_COMPRESSION_RATIO Ratio of uncompressed bytes (in-memory) to compressed bytes (on-disk) for all nodes.

TOKUDB_PIVOTS_FETCHED_FOR_PREFETCH_BYTES Number of bytes of pivot nodes fetched by a prefetch thread.

TOKUDB_PIVOTS_FETCHED_FOR_PREFETCH Number of pivot nodes fetched by a prefetch thread.

TOKUDB_PIVOTS_FETCHED_FOR_PREFETCH_SECONDS Number seconds waiting for IO when fetching pivot nodes by a prefetch thread.

TOKUDB_PIVOTS_FETCHED_FOR_QUERY_BYTES Number of bytes of pivot nodes fetched for queries.

TOKUDB_PIVOTS_FETCHED_FOR_QUERY Number of pivot nodes fetched for queries.

TOKUDB_PIVOTS_FETCHED_FOR_QUERY_SECONDS Number of seconds waiting for IO when fetching pivot nodes for queries.

TOKUDB_PIVOTS_FETCHED_FOR_WRITE_BYTES Number of bytes of pivot nodes fetched for writes.

TOKUDB_PIVOTS_FETCHED_FOR_WRITE Number of pivot nodes fetched for writes.

TOKUDB_PIVOTS_FETCHED_FOR_WRITE_SECONDS Number of seconds waiting for IO when fetching pivot nodes for writes.

TOKUDB_PROMOTION_H1_ROOTS_INJECTED_INTO Number of times a message stopped at a root with height 1.

TOKUDB_PROMOTION_INJECTIONS_AT_DEPTH_0 Number of times a message stopped at depth 0.

TOKUDB_PROMOTION_INJECTIONS_AT_DEPTH_1 Number of times a message stopped at depth 1.

TOKUDB_PROMOTION_INJECTIONS_AT_DEPTH_2 Number of times a message stopped at depth 2.

TOKUDB_PROMOTION_INJECTIONS_AT_DEPTH_3 Number of times a message stopped at depth 3.

TOKUDB_PROMOTION_INJECTIONS_LOWER_THAN_DEPTH_3 Number of times a message was promoted past depth 3.

TOKUDB_PROMOTION_LEAF_ROOTS_INJECTED_INTO Number of times a message stopped at a root with height 0.

TOKUDB_PROMOTION_ROOTS_SPLIT Number of times the root split during promotion.

TOKUDB_PROMOTION_STOPPED_AFTER_LOCKING_CHILD Number of times a message stopped before a child which had been locked.

TOKUDB_PROMOTION_STOPPED_AT_HEIGHT_1 Number of times a message stopped because it had reached height 1.

TOKUDB_PROMOTION_STOPPED_CHILD_LOCKED_OR_NOT_IN_MEMORY Number of times a message stopped because it could not cheaply get access to a child.

TOKUDB_PROMOTION_STOPPED_CHILD_NOT_FULLY_IN_MEMORY Number of times a message stopped because it could not cheaply get access to a child.

TOKUDB_PROMOTION_STOPPED_NONEMPTY_BUFFER Number of times a message stopped because it reached a nonempty buffer.

TOKUDB_TXN_ABORTS This is the total number of transactions that have been aborted.

TOKUDB_TXN_BEGIN This is the number of transactions that have been started.

TOKUDB_TXN_BEGIN_READ_ONLY Number of read only transactions started.

TOKUDB_TXN_COMMITS This is the total number of transactions that have been committed.

CHAPTER

SEVEN

APPENDIX

7.1 Fast Upserts and Updates

Upserts (`INSERT ... ON DUPLICATE KEY UPDATE`) and Updates are slow because they are implemented as read row, modify row, write row. This implementation is needed because the number of rows affected by the update is returned, and the computation of affected rows requires one to know whether or not the row existed before the update or upsert operation. It also limits the throughput of these statements to read IO performance of the system.

On workloads where computing the number of affected rows is not important it is possible to execute these statements with no read requirement. The remainder of this section describes the conditions under which this optimization applies.

7.1.1 Example Tables

Table with a simple primary key:

```
CREATE TABLE table (
  id INT NOT NULL,
  column BIGINT NOT NULL,
  PRIMARY KEY(id)) ENGINE=TokuDB;
```

Table with a compound primary key:

```
CREATE TABLE table (
  id_1 INT NOT NULL,
  id_2 BIGINT NOT NULL,
  column_int INT UNSIGNED NOT NULL,
  column_char CHAR(32),
  column_var VARCHAR(32),
  PRIMARY KEY(id_1,id_2)) ENGINE=TokuDB;
```

7.1.2 Currently Supported Update Expressions

Set a char field to a constant string:

```
column_char = 'Greetings!'
```

Set an integer field to a constant:

```
column_int = 42
```

Add a constant to an integer field:

```
column_int = column_int + 100
```

Subtract a constant from an integer field:

```
column_int = column_int + 1
```

Decrement an integer field if its value is not already zero:

```
column_int = if ( column_int = 0 , column_int - 1 )
```

More supported expressions will be added over time.

7.1.3 Example Upserts

Insert a row. If the row already exists, then increment one of its columns.

```
INSERT NOAR INTO table
  VALUES (1000,0)
  ON DUPLICATE KEY UPDATE column_int = column_int + 1;
```

Insert a row. If the row already exists, then increment one of its columns, and set another of its columns.

```
INSERT NOAR INTO table
  VALUES (2000,3000,0,'Greetings!','Salutations!')
  ON DUPLICATE KEY UPDATE column_int = column_int + 1,
  column_char = 'Greetings and Salutations!';
```

7.1.4 Requirements for Fast Upserts

- The `NOAR` keyword is required. It stands for "no affected rows". Its presence forces the TokuDB storage engine to perform a fast operation or fail if it is unable to due to any of the following requirements not being met.
- The table must have a defined primary key.
- The table cannot have any secondary keys.
- No triggers on the table.
- If binary logging is enabled, then `binlog_format` must be set to `STATEMENT`.
- The update fields must be int, char, or varchar types.
- Session must be in relaxed mode
  - `SQL_MODE` not set for `STRICT_ALL_TABLES` and `STRICT_TRANS_TABLES`.
  - Numeric values are clipped at their maximum value.

7.1.5 Example Updates

Set a column to a constant for the row with primary key 42. If the row does not exist, then nothing is changed.

```
UPDATE NOAR table
  SET column = 1
  WHERE id = 42;
```

Increment a column for the row with primary key 100. If the row does not exist, then nothing is changed.

```
UPDATE NOAR table
  SET column = column + 1
  WHERE id = 100;
```

Decrement a column x and set column c to a constant string for the row with primary key ida 7 and idb 8. If the row does not exist, then nothing is changed.

```
UPDATE NOAR table
  SET column_int = column_int - 1,
  column_char = 'Greetings!'
  WHERE id_1 = 7 AND id_2 = 8;
```

7.1.6 Requirements for Fast Updates

- The `NOAR` keyword is required and stands for "no affected rows". Its presence forces the TokuDB storage engine to perform a fast operation or fail if it is unable to due to any of the following requirements not being met.
- The table must have a defined primary key.
- The primary key fields must have int, char or varchar type.
- Updated columns cannot be part of any secondary keys.
- The table cannot have any clustering keys.
- The `WHERE` condition must resolve to a single row.
- No triggers on the table.
- If binary logging is enabled, then `binlog_format` must be set to `STATEMENT`.
- The update fields must be int, char, or varchar types.
- Session must be in relaxed mode
  - `SQL_MODE` not set for `STRICT_ALL_TABLES` and `STRICT_TRANS_TABLES`.
  - Numeric values are clipped at their maximum value.

7.2 Compiling MySQL from Source

It is not necessary to build MySQL and the TokuDB for MySQL® handlerton 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 MySQL 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 MySQL that is based on the MySQL 5.5.30 source.

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

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

7.2.1 System and Hardware Requirements

Operating Systems: These instructions were tested using CentOS 5.8. 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:

- autoconf 2.61
- automake 1.10
- bison 2.3
- ccache 2.4
- cmake 2.6
- libtool 1.5
- ncurses-devel 5.6
- readline 5.2
- zlib-devel 1.2

Processor Architecture: TokuDB requires a 64-bit operating system

Disk space: 1 GB

7.2.2 Download and verify files

Download and verify the following files from the Tokutek web site and put them in the build directory:

- `mysql-5.5.30-tokudb-7.1.0-0-src.tar.gz`
- `mysql-5.5.30-tokudb-7.1.0-0-src.tar.gz.md5`
- `tokufractaltreeindex-7.1.0-0-linux-x86 64.tar.gz`
- `tokufractaltreeindex-7.1.0-0-linux-x86 64.tar.gz.md5`

7.2.3 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.

1. Extract the `tokudb.build.bash` script from the source tarball

   ```
   $ tar xzf mysql-5.5.30-tokudb-7.1.0-0-src.tar.gz
   ```

2. Move the script to the current directory:

   ```
   $ mv mysql-5.5.30-tokudb-7.1.0-0-src/scripts/tokudb.build.bash .
   ```

3. Make the script executable:

   ```
   $ chmod 744 tokudb.build.bash
   ```

4. Build the binary release tarball:

   ```
   ./tokudb.build.bash
   ```

7.2.4 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.

7.3 3rd Party Libraries

7.3.1 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:

- Redistributions of source code must retain the above copyright notice(s), this list of conditions and the following disclaimer.
- 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.

CHAPTER

EIGHT

RELEASE NOTES

8.1 TokuDB 7.x

8.1.1 Version 7.5.5

New Features

- Rebase onto MySQL 5.5.41
- Rebase onto MariaDB 5.5.41
- Added the ability to throttle optimize table
- Added the ability to optimize individual indexes of a table
- Added the ability to optimize a percentage of a fractal tree
- Enhanced lock timeout debugging by logging both the blocked and blocking sql
- Make alter table reorganize partition interruptable
- Hot Backup is now implemented as a plugin
- Add debug info to records_in_range to help debug query plan problems
- Added a variable to disable the read only slave check

Bug Fixes

- Fix a MySQL bug when updating a partitioned table

Upgrade Considerations

- The hot backup syntax has changed.
- Upgrade after a dirty shutdown of TokuDB 7.1.5+ is supported. Upgrading from any version prior to 7.1.5 requires a clean shutdown

8.1.2 Version 7.5.3

New Features

- Rebase onto MySQL 5.5.40
- Rebase onto MariaDB 5.5.40

Bug Fixes

- Fix crash in TC_LOG_MMAP::get_active_from_pool

- Fix TokuDB info schema error handling in MariaDB 10
- Fix cardinality data for partitioned TokuDB tables
- Fix alter table analyze partition for TokuDB tables
- Fix issue where add/drop index may break TokuDB cardinality data

Upgrade Considerations

- TokuDB 7.5.3 uses file format version 27
- Upgrade after a dirty shutdown of TokuDB 7.1.5+ is supported. Upgrading from any version prior to 7.1.5 requires a clean shutdown

8.1.3 Version 7.5.2

Bug Fixes

- Removed overlocking caused by before/after insert/update/delete triggers on primary key ranges
- Fixed issue where unique key violations were falsely reported

Upgrade Considerations

- TokuDB 7.5.2 uses file format version 27
- Upgrade after a dirty shutdown of TokuDB 7.1.5+ is supported. Upgrading from any version prior to 7.1.5 requires a clean shutdown

8.1.4 Version 7.5.0

New Features

- Created capability for TokuDB replication slave servers to avoid IO for uniqueness checks and update/delete statements
- Added support for multiple source directories in hot backup (enterprise feature)
- Added bulk fetch algorithm to improve performance of select and delete statements
- Improved output of TokuDB information schema plugins, (now includes database, table, and index names separately)
- Rebase onto MySQL 5.5.39
- Rebase onto MariaDB 5.5.39
- Several Fractal Tree index improvements
- Fixed bug in XA recovery
- Added configuration parameter to avoid fsync() in XA prepare phase
- Upgraded jemalloc to version 3.6.0

Upgrade Considerations

- TokuDB 7.5.0 uses file format version 27
- Upgrade after a dirty shutdown of TokuDB 7.1.5+ is supported. Upgrading from any version prior to 7.1.5 requires a clean shutdown

8.1.5 Version 7.1.8

TokuDB 7.1.8 was an unreleased version that contains several bug fixes.

8.1.6 Version 7.1.7

Upgrade Considerations

- TokuDB 7.1.7 uses file format version 25.
- Upgrade after a dirty shutdown of TokuDB 7.1.5+ is supported. Upgrading from any version prior to 7.1.5 requires a clean shutdown.

MariaDB

- Rebase onto MariaDB 5.5.38

MySQL

- Rebase onto MySQL 5.5.38

MySQL and MariaDB

- Hot optimize for MySQL 5.6 and MariaDB 10.0 using alter recreate
- Use thd_ha_data and thd_ha_set_data functions
- Fail TokuDB initialization if jemalloc is not loaded
- Hot backup direct I/O fix
- Speed up unique key check after load
- Use row estimate to decide if bulk loader is used to insert rows
- TokuDB sets invalid proc info
- MDEV-6324 uninitialied var in tokudb discover3 function

Fractal Tree index changes

- Fix some TokuDB loader bugs when NPROC limit exceeded
- Fix the dbremove assert when NPROC limit exceeded
- Fix the thread pool assert when NPROC limit exceeded
- Fix a TokuDB loader bug that would unlink the in use fractal tree files
- Fix a file descriptor leak in the bulk loader

8.1.7 Version 7.1.6

Upgrade Considerations

- TokuDB 7.1.6 uses file format version 25. Upgrade after a dirty shutdown of TokuDB 7.1.5 is supported. Upgrade after a clean shutdown of versions prior to TokuDB 7.1.5 is required.
- TokuDB 7.1.5 uses file format version 25. Upgrade after a clean shutdown of prior versions is required.
- TokuDB 7.1.0 through TokuDB 7.0.1 use file format version 24. Upgrade after a clean shutdown of versions prior to TokuDB 7.0 is required.

MariaDB

- Rebase onto MariaDB 5.5.37

MySQL

- Rebase onto MySQL 5.5.37
- RQG temporal replication caused crash in TokuDB

MySQL and MariaDB

- Replace into replication bug
- Unique secondary key overlocking
- MDEV-5646 analyze table may crash
- MDEV-5932 wrong result for create table select
- Open table stall due to lots of deletes in the tree
- SQL Bench test insert performance regression
- MDEV-4533 delete ignore and row replication
- i_s_tokudb_lock_waits_released test sometimes fails
- MDEV-5595 slow point query with deleted key
- MDEV-6162 Incorrect use of rw locks

Fractal Tree index changes

- Change table open cost from O(n) to O(log n)
- Bound point query Fractal Tree index search
- Fix impossibly long index creation time due to deleted rows at the left edge of the tree

8.1.8 Version 7.1.5

Upgrade considerations

- Note that TokuDB v7.1.5 includes file format changes and will automatically upgrade prior versions of TokuDB. However, this is a one-way process and prior versions of TokuDB are not compatible with newer versions. If you require the ability to revert to an older version of TokuDB you must make a full backup of your data folder prior to running this new version.

MariaDB

- Rebase onto MariaDB 5.5.36
  - Fix MDEV-5458
  - Fix MDEV-5404
  - Fix MDEV-5396
  - Fix MDEV-5405
- Fix memory leak on view
- Fix handlersocket library dependency

MySQL

- Rebase onto MySQL 5.5.36
- Fix last lock timeout memory leak

MariaDB and MySQL

- Use Red Hat devtoolset-1.1 (with its gcc 4.7) to build mysql and mariadb1,2
- Blocked lock tree requests are not killable
- Get rid of metadata_db and tokudb_meta_mutex to remove causes of stalls during table open
- Update cardinality only on first table open to avoid possible lock tree race
- `Field::null_ptr` is not set properly, wrong result on a NOT IN subquery with NULL values, also known as MDEV-5399
- TokuDB crash after cancel of online create index, now it won't start
- Optimize temporary table crash,2
- Fix race in tokudb.i_s_tokudb_lock_waits_releases test,2
- HCAD update function does not initialize the entire update message
- uint3korr reads beyond the end of the buffer
- `tokudb::analyze` conditional jump uninitialized value
- tokudb leaks dbt arrays
- tokudb should destroy the tokudb_primary_key_bytes_inserted partitioned counter
- Cleanup use of thd_proc_info to not reference stack variables after function returns
- Alter table sometimes ignores a change in old and new null bytes which leads to broken row encoding
- Enum values not being actualized after alter table
- Alter table drop primary key fails,2,3
- Zero length character columns cause drop column to assert
- Uniqueness violation during alter table rebuild gets the -100010 error
- show create table does not always display the auto_increment value
- Slave exec mode idempotent is broken
- Fix `tokudb::index_next_same` to set table->status
- Alter table operations (only partition related?) corrupt nullable columns bitmap
- Cleanup mysql-test files with Windows style line endings
- Map tokudb_small to lzma and tokudb_fast to quicklz
- Use session variable to control lock timeout
- Use session variable to control loader memory size
- RQG tests percona_qa/percona_qa.yy and temporal/current_timestamp_6.yy cause a crash on cachetable,2

Several changes to Fractal Tree index, the highlights of which include:

- Run full garbage collection after injecting into an overfull leaf node
- Avoid stalling small transactions when lock escalation is running,2
- Mempool can grow uncontrollably if it is long lived
- Continuous checkpointing can starve a hot index operation

8.1.9 Version 7.1.0

Features:

- Added ability for users to view lock information via information_schema.tokudb_trx, information_schema.tokudb_locks, and information_schema.tokudb_lock_waits_tables. More information is available in Appendix B.
- Changed default compression to zlib, tokudb_row_format=tokudb_zlib.
- Changed default basement node size to 64K, tokudb_read_block_size=65536.
- Changed default analyze time to 5 seconds, tokudb_analyze_time=5.
- Changed default loader behavior to compress intermediate files, tokudb_load_save_space=ON.
- Added server variable to control amount of memory allocated for each bulk loader, tokudb_loader_memory_size. In prior TokuDB versions each loader allocated 50% of the available TokuDB cache.
- Added session variable to control output of lock timeouts, tokudb_lock_timeout_debug.
- Added session variable to hold information relating to the last lock timeout for the current session, tokudb_last_lock_timeout.
- Added plug-in tokudb_trx.
- Added plug-in tokudb_locks.
- Added plug-in tokudb_lock_waits.
- Removed plug-in tokudb_user_data.
- Removed plug-in tokudb_user_data_exact.
- Changed table close behavior such that all data for the table remains in the cache (and is not flushed immediately)
- Changed checkpointing behavior such that the next checkpoint now begins tokudb_checkpointing_period seconds after the prior checkpoint started, not after it completed.

Bug fixes:

- Fixed bug where keys containing case-insensitive fields may not be abortable (Tokutek/ft-engine/issues/94).
- Fixed several stalls (Tokutek/ft-engine/issues/95, Tokutek/ft-engine/issues/73, Tokutek/ft- engine/issues/66).
- Fixed issue where converting InnoDB tables with key_block_size created TokuDB tables with small basement node sizing (Tokutek/ft-engine/issues/62).
- Backported MariaDB processlist “huge time” bug (Tokutek/mariadb/9).

8.1.10 Version 7.0.4

Bug fixes:

- Disabled hot column expansion of text and blob data types. Expanding these types must be performed via a table rewrite by setting tokudb_disable_hot_alter=ON for the session.

8.1.11 Version 7.0.3

Features:

- Improved performance of in-memory point queries and secondary index range queries (ft-index #5).

- Allow users to control fsync frequency via tokudb fsync log period server variable (#47).
- Server crash uses gdb (if available) to produce additional trace information for support (#49).

Bug fixes:

- "1034 Incorrect key file" issue (#52)
- Performance regression in sql-bench/wisconsin test (#51)
- Performance regression in iiBench (#48)
- Added plug-in to MTRv1 (#46).
- Return error if we try to create a table with a deprecated field type (#42).
- Server crash when open file limit reached (#30).
- Overactive assert in ha `tokudb::external` lock (#2).

8.1.12 Version 7.0.2

Enterprise Edition released with support for online backup.

8.1.13 Version 7.0.1

MySQL 5.5 and MariaDB 5.5 Improvements.

- Added support for Direct IO.
- Improved cardinality accounting for query planning.
- Improved row estimates for query planning.
- Read-only transaction performance improvements.

MariaDB 5.5 Improvements

- Added support for Extended Keys
- Added support for Index Condition Pushdown (ICP)
- Added support for Multi-Range Reads (MRR)

Fast-Updates Improvements

- Added NOAR syntax
- Added support for Statement Based Replication
- Added support for update of VARCHAR columns

Additional changes include:

- Modified cachetable eviction algorithm to favor ejection of large objects
- Added most of TokuDB engine status information to information schema.global status
- Improved shutdown time on systems with lots of tables
- Added detection of transparent huge pages (server shuts down)

8.2 TokuDB 6.x

8.2.1 Version 6.6.7

Bug fixes:

- Improved row estimates if within single basement node (#6285)
- Lock escalation when transactions relock (#6283)
- Splitting empty leaf nodes crashed server if also rebalancing (#6300)
- Race condition when multiple threads performing locks and unlocks on high concurrency workloads (#5979)

8.2.2 Version 6.6.5

Bug fixes:

- INSERT ... SELECT statements were grabbing locks when isolation level ¡= READ COMMITTED (#5974)
- Lock escalation was not applied to transaction range buffers, causing large RSS (#5977)

8.2.3 Version 6.6.4

Bug fixes:

- Fixed upgrade bug preventing multiple upgrades after v6.0 (#5902)

8.2.4 Version 6.6.3

Changes include:

- Added optimization for certain UPDATE and INSERT ON DUPLICATE KEY UPDATE statements.
- Significant performance improvements for single threaded and concurrent workloads, both in- memory and larger than RAM.
- Auto-increment column now resets to default value on truncate table.
- MySQL 5.1 and MariaDB 5.2 are no longer supported.
- Bulk loader now optionally compresses intermediate files.
- Added new information_schema plugins tokudb_fractal_tree_block_map and tokudb_fractal_tree_info.
- Updated MariaDB 5.5 version to 5.5.28a
- Updated MySQL 5.5 version to 5.5.28
  - Added fix for security issue CVE-2012-5579 in MySQL 5.5.28 binaries (fix from MariaDB 5.5.28a).

8.2.5 Version 6.5.1

Bug fixes:

- Hot Column Expand operation on a table with deleted rows could cause server crash (#5674)
- Simultaneously dropping an index and querying using the index could cause server crash (#5679)

8.2.6 Version 6.5.0

Changes include:

- Hot column expansion for varchar, char, varbinary, and integer data types
- Support for hot operations on partitioned tables
- Lock and latch refinement in the cachetable
- QPS and concurrency improvements for reads on in-memory workloads

8.2.7 Version 6.1.1

Bug fixes:

- Auto-upgraded tables (prior to v6.1.0) with significant insertions using auto-increment could cause server crash (#5435)
- HCR operations could starve on MySQL 5.5 and MariaDB 5.5 builds on busy servers (#5377)
- Dynamically changing row format on MariaDB 5.2 and MariaDB 5.5 builds could cause server crash (#5429)

8.2.8 Version 6.1.0

Changes include:

- TokuDB is now the default storage engine
- sql_mode now defaults to NO_ENGINE_SUBSTITUTION, an error is returned if the requested storage engine is unavailable
- Added HCAD support to MySQL 5.5 version
- Updated MySQL 5.5 version to 5.5.24
- Added support for MariaDB 5.5 (5.5.25)
- Improved in-memory point query performance via lock/latch refinement
- Updated jemalloc to version 3
- Removed requirement that LD_PRELOAD and LD_LIBRARY_PATH be set prior to starting mysqld.
- Modifying table compression (tokudb_row_format) is now a hot operation
- Modifying auto_increment value is now a hot operation
- Added TokuDB_file_map plug-in to show relationship between indexes and files

8.2.9 Version 6.0.1

Features:

- Auto-increment values can now be modified (the table is recreated via a slow alter operation)
- Compression type can now be modified (the table is recreated via a slow alter operation)

Bug fixes:

- Patched MySQL authentication bug into all builds (#5079)
- Fixed MySQL bug where slave could crash on XA transactions spanning multiple storage engines (#5001)

- Fixed server crash on the following conditions:
  - Insert ignore scenario into empty table (#5003)
  - Insert scenario into particular schema under MariaDB 5.2.10 (#5013)
  - Race condition in upgrade/checkpointing scenario (#5016)
  - Auto-upgrading MySQL 5.5 database with existing unapplied HCAD messages (#5023)

8.2.10 Version 6.0.0

Changes include:

- Reduced checkpoint variability
- Performance enhancements for multi-client workloads
- Performance enhancements for in-memory workloads
- Added support for LZMA compression algorithm
- Added capability to define compression algorithm via DDL or the tokudb_row_format session variable
- Added ability to suppress client error messages in the server log via the tokudb_log_client_errors session variable
- Added support for XA
- Added compatibility with prior versions of TokuDB (4.2.x, 5.0.x, 5.2.x)
- Added support for MySQL 5.5
- Updated MySQL 5.1 version to 5.1.61
- Updated MariaDB 5.2 version to 5.2.10

8.3 TokuDB 5.x

8.3.1 Version 5.2.7

Changes include:

- 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.

8.3.2 Version 5.0.6

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.

8.3.3 Version 5.0.5

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.

8.3.4 Version 5.0.4

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 MySQL Bug 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.

8.3.5 Version 5.0.3

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 up- dates. Once the index creation completes, it is 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;`

8.4 TokuDB 4.x

8.4.1 Version 4.2.0

Changes include:

- 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.

8.4.2 Version 4.1.1

Bug fixes:

- Fixed a bug in 4.1.0.

8.4.3 Version 4.1.0

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.

8.4.4 Version 4.0.0

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.

Features:

- READ COMMITTED isolation level.

8.5 TokuDB 3.x

8.5.1 Version 3.1.0

Features:

- 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.

8.5.2 Version 3.0.4

Changes include:

- 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.

8.5.3 Version 3.0.3

Changes include:

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

8.5.4 Version 3.0.2

Changes include:

- 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.

8.5.5 Version 3.0.1

Changes include:

- 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.

8.5.6 Version 3.0.0

Changes include:

- 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.

8.6 TokuDB 2.x

8.6.1 Version 2.2.0

Changes include:

- Increase in multi-threaded 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:
  - `LOAD DATA INFILE...`
  - `INSERT INTO table SELECT *...`

8.6.2 Version 2.1.0

Changes include:

- 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.

8.6.3 Version 2.0.2

Changes include:

- 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.

8.6.4 Version 2.0.1

Changes include:

- Crash safety for Windows.
- CREATE INDEX is now abort-able. 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.

8.6.5 Version 2.0.0

Changes include:

- 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.

CHAPTER

NINE

GETTING THE MOST FROM TOKUDB

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 up to 25x for high compression. Compression can also lead to improved performance since less data needs to be read from and written to disk.

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.

Eliminates Slave Lag: TokuDB replication slaves can be configured to process the replication stream with virtually no read IO. Uniqueness checking is performed on the TokuDB master and can be skipped on all TokuDB slaves. Also, row based replication ensures that all before and after row images are captured in the binary logs, so the TokuDB slaves can harness the power of Fractal Tree indexes and bypass traditional read-modify-write behavior. This "Read Free Replication" ensures that replication slaves do not fall behind the master and can be used for read scaling, backups, and disaster recovery, without sharding, expensive hardware, or limits on what can be replicated.

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 MySQL 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, Expansion and Rename: TokuDB allows the addition of new columns to an existing table, the deletion of existing columns from an existing table, the expansion of char, varchar, varbinary, and integer type columns in an existing table, and the renaming of an existing column while inserts and queries are being performed on that table.

Online (Hot) Backup: The TokuDB Enterprise Edition can create backups of online database servers without down-time.

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. Thus, fast indexing means fast queries.

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.

Less Aging/Fragmentation: TokuDB can run much longer, likely indefinitely, without the need to perform the customary practice of dump/reload or `OPTIMIZE TABLE` to restore database performance. The key is the fundamental difference with which the Fractal Tree stores data on disk. Since, by default, the Fractal Tree will store data in 4MB chunks (pre-compression), as compared to InnoDB's 16KB, TokuDB has the ability to avoid "database disorder" up to 250x better than InnoDB.

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.

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

Lock Diagnostics: TokuDB provides users with the tools to diagnose locking and deadlock issues. For more information, see *Lock Visualization in TokuDB*.

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.

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