TokuDB Troubleshooting

March, 22, 2017

Sveta Smirnova, George O. Lorch III, Vlad Lesin

Table of Contents

PERCONA

What you need to know about TokuDB?

Full featured engine

- Transactional
- ACID
- MVCC
- Row-level locking
- XA
- Transaction Isolation Levels
- No Foreign Keys

PERCONA

Optimized for Writes

- Fast inserts
- Bulk loader
- Compression
- Fractal Tree

PERCONA

Optimizations for Retrievals

- Clustered secondary indexes
- Indexes can include up to 32 columns
- Indexes do not age

PERCONA

Online Operations

- OPTIMIZE TABLE
- Index creations
  - DROP INDEX is offline
- Hot column operations
  - Addition
  - Deletion
  - Expansion
  - Rename
- Backup

PERCONA

Read Free Replication

- Slave-only feature
  - Requires RBR on master
- Fast slaves
- Must be read-only
- Tables must have UNIQUE key
- You can use any engine on master

PERCONA

Powerful Diagnostic Tools

- SHOW ENGINE TOKUDB STATUS
- Detailed lock information
- Tables in Information Schema
- No Performance Schema Support
  - Runtime statistics in SHOW ENGINE TOKUDB STATUS
  - Will be added in future versions

PERCONA

Data Corruption and Inconsistencies

How TokuDB Deals with Data Corruption?

- Handled automatically at startup
  - Fixes known inconsistencies
  - Real corruptions are not reparable!

PERCONA

How TokuDB Deals with Data Corruption?

- Handled automatically at startup
- You interfere this process only slightly
  - There is no option, similar to innodb_force_recovery
  - Feature Request FT-733
  - tokudb_strip_frm_data removes embedded .frm metadata from status files
    - Should be used with great care!

PERCONA

How TokuDB Deals with Data Corruption?

- Handled automatically at startup
- You interfere this process only slightly
- Backup restoration required when corruption cannot be cured automatically

PERCONA

How TokuDB Deals with Data Corruption?

- Handled automatically at startup
- You interfere this process only slightly
- Backup restoration required when corruption cannot be cured automatically
- Disable write cache for safety

PERCONA

How TokuDB Deals with Data Corruption?

- Handled automatically at startup
- You interfere this process only slightly
- Backup restoration required when corruption cannot be cured automatically
- Disable write cache for safety
- Verify your backups!

PERCONA

Corruptions vs Inonsistencies

- What are Symptoms of Real Corruption?
  - Assertions about bad checksums
  - Assertions while unpaking row data
  - Not recoverable

PERCONA

Corruptions vs Inonsistencies

- What are Symptoms of Real Corruption?
  - Assertions about bad checksums
  - Assertions while unpaking row data
  - Not recoverable
- What are Symptoms of Inconsistency?
  - Different query results for Primary vs secondary key access?

PERCONA

Automatic Inconsistencies Recovery

- Are inconsistency limited to know key range?
  - Recoverable

PERCONA

Automatic Inconsistencies Recovery

- Are inconsistency limited to know key range?
- Is it not possible to limit inconsistency to known range?
  - Not recoverable
  - Use dump-reload

PERCONA

Automatic Inconsistencies Recovery

- Are inconsistency limited to know key range?
- Is it not possible to limit inconsistency to known range?
- There can be some data "loss" or "gain"
  - If Primary key misses data
  - If Primary key still contains "deleted" rows

PERCONA

File Formats

- TokuDB file formats are not the same across MySQL variants
  - Percona Server 5.6 and 5.7
  - MariaDB 10.x
  - Builds, released by Tokutek
    - On Oracle MySQL
    - On MariaDB

PERCONA

File Formats

- TokuDB file formats are not the same across MySQL variants
  - Percona Server 5.6 and 5.7
  - MariaDB 10.x
  - Builds, released by Tokutek
    - On Oracle MySQL
    - On MariaDB
- Migrating will require logical dump and reload
  - Upgrade from Percona Server 5.6 to Percona Server 5.7 can be in place

TokuDB Hotbackup

- Can cause InnoDB corruptions with innodb_use_native_aio=1

PERCONA

TokuDB Hotbackup

- Can cause InnoDB corruptions with innodb_use_native_aio=1
- Cannot be used to create new slave
  - Work in progress to solve this

PERCONA

Locks

Locks in TokuDB

- Row-level
- No Auto Increment lock tuning
- Table locks for table definition modifications

PERCONA

Which Issues Will We Discuss?

- Lock wait timeout
  - `session1> START TRANSACTION;`

PERCONA

Which Issues Will We Discuss?

- Lock wait timeout
  - `session1> START TRANSACTION;`
  - `session1> UPDATE t SET f='foo' WHERE id=12345;`

PERCONA

Which Issues Will We Discuss?

- Lock wait timeout
  - `session1> START TRANSACTION;`
  - `session1> UPDATE t SET f='foo' WHERE id=12345;`
  - `session2> UPDATE t SET f='bar' WHERE id=12345;`

PERCONA

Which Issues Will We Discuss?

- Lock wait timeout
- Deadlocks
  - `session1> START TRANSACTION;`

PERCONA

Which Issues Will We Discuss?

- Lock wait timeout
- Deadlocks
  - `session1> START TRANSACTION;`
  - `session1> SELECT * FROM t WHERE id=12345 FOR UPDATE;`

PERCONA

Which Issues Will We Discuss?

- Lock wait timeout
- Deadlocks
  - `session1> START TRANSACTION;`
  - `session1> SELECT * FROM t WHERE id=12345 FOR UPDATE;`
  - `session2> START TRANSACTION;`

PERCONA

Which Issues Will We Discuss?

- Lock wait timeout
- Deadlocks
  - `session1> START TRANSACTION;`
  - `session1> SELECT * FROM t WHERE id=12345 FOR UPDATE;`
  - `session2> START TRANSACTION;`
  - `session2> SELECT * FROM t WHERE id=54321 FOR UPDATE;`

PERCONA

Which Issues Will We Discuss?

- Lock wait timeout
- Deadlocks
  - `session1> START TRANSACTION;`
  - `session1> SELECT * FROM t WHERE id=12345 FOR UPDATE;`
  - `session2> START TRANSACTION;`
  - `session2> SELECT * FROM t WHERE id=54321 FOR UPDATE;`
  - `session1> UPDATE t SET f='foo' WHERE id=54321;`

PERCONA

Which Issues Will We Discuss?

- Lock wait timeout
- Deadlocks
  - `session1> START TRANSACTION;`
  - `session1> SELECT * FROM t WHERE id=12345 FOR UPDATE;`
  - `session2> START TRANSACTION;`
  - `session2> SELECT * FROM t WHERE id=54321 FOR UPDATE;`
  - `session1> UPDATE t SET f='foo' WHERE id=54321;`
  - `session2> UPDATE t SET f='foo' WHERE id=12345;`

PERCONA

Locks Diagnostics

- Information Schema tables
  - TOKUDB_TRX
  - TOKUDB_LOCKS
  - TOKUDB_LOCK_WAITS

PERCONA

Locks Diagnostics

- Information Schema tables
- Variable tokudb_lock_timeout_debug
  - Controls how lock timeouts and deadlocks are reported
    - 0 - Nothing reported
    - 1 - JSON in tokudb_last_lock_timeout variable
    - 2 - JSON in error log
    - 3 - JSON in tokudb_last_lock_timeout and error log
  - Session variable!

PERCONA

Locks Diagnostics

- Information Schema tables
- Variable tokudb_lock_timeout_debug
- Variable tokudb_last_lock_timeout
  - Contains JSON document, described latest lock conflict for the session

PERCONA

By example: lock wait timeout

- Sessions

PERCONA

By example: lock wait timeout

- Sessions

```
session1> START TRANSACTION;
session1> UPDATE t SET f='foo' WHERE id=12345;
```


PERCONA

By example: lock wait timeout

- Sessions

```
session1> START TRANSACTION;
session1> UPDATE t SET f='foo' WHERE id=12345;

session2> UPDATE t SET f='bar' WHERE id=12345;
ERROR 1205 (HY000): Lock wait timeout exceeded; try restarting transaction
```


PERCONA

By example: lock wait timeout

- Sessions

```
session1> START TRANSACTION;
session1> UPDATE t SET f='foo' WHERE id=12345;

session2> UPDATE t SET f='bar' WHERE id=12345;
ERROR 1205 (HY000): Lock wait timeout exceeded; try restarting transaction

session1> select @@tokudb_last_lock_timeout\G
*************************** 1. row ***************************
@@tokudb_last_lock_timeout: NULL
1 row in set (0.00 sec)
```


PERCONA

By example: lock wait timeout

- Sessions

```
session1> START TRANSACTION;
session1> UPDATE t SET f='foo' WHERE id=12345;

session2> UPDATE t SET f='bar' WHERE id=12345;
ERROR 1205 (HY000): Lock wait timeout exceeded; try restarting transaction

session1> select @@tokudb_last_lock_timeout\G
*************************** 1. row ***************************
@@tokudb_last_lock_timeout: NULL
1 row in set (0.00 sec)

session2> select @@tokudb_last_lock_timeout\G
*************************** 1. row ***************************
@@tokudb_last_lock_timeout: {"mysql_thread_id":336, "dbname":"./test/t-main",
"requesting_txnid":16, "blocking_txnid":13,
"key_left":"ff39300000", "key_right":"0139300000"}
1 row in set (0.00 sec)
```

PERCONA

By example: lock wait timeout

- Sessions
- Error log

```
2017-03-12T13:18:17.758148Z 336 [ERROR] TokuDB: lock timeout
{"mysql_thread_id":336, "dbname":"./test/t-main", "requesting_txnid":48, "blocking_txnid":13,
"key_left":"ff39300000", "key_right":"0139300000"}
2017-03-12T13:18:17.758206Z 336 [ERROR] TokuDB:
requesting_thread_id:336 q:UPDATE t SET f='bar' WHERE id=12345
```

PERCONA

By example: Deadlock

- Sessions

```
session1> START TRANSACTION;
session1> SELECT * FROM t WHERE id=12345 FOR UPDATE;
```


PERCONA

By example: Deadlock

- Sessions

```
session1> START TRANSACTION;
session1> SELECT * FROM t WHERE id=12345 FOR UPDATE;

session2> START TRANSACTION;
session2> SELECT * FROM t WHERE id=54321 FOR UPDATE;
```


PERCONA

By example: Deadlock

- Sessions

```
session1> START TRANSACTION;
session1> SELECT * FROM t WHERE id=12345 FOR UPDATE;

session2> START TRANSACTION;
session2> SELECT * FROM t WHERE id=54321 FOR UPDATE;

session1> UPDATE t SET f='foo' WHERE id=54321;
```

PERCONA

By example: Deadlock

- Sessions

```
session1> START TRANSACTION;
session1> SELECT * FROM t WHERE id=12345 FOR UPDATE;

session2> START TRANSACTION;
session2> SELECT * FROM t WHERE id=54321 FOR UPDATE;

session1> UPDATE t SET f='foo' WHERE id=54321;

session2> UPDATE t SET f='foo' WHERE id=12345;
ERROR 1213 (40001): Deadlock found when trying to get lock; try restarting transaction
session2> select @@tokudb_last_lock_timeout\G
*************************** 1. row ***************************
@@tokudb_last_lock_timeout: {"mysql_thread_id":2688, "dbname":"./test/t-main",
"requesting_txnid":80, "blocking_txnid":78, "key_left":"ff39300000",  "key_right":"0139300000"}
1 row in set (0.00 sec)
```

PERCONA

By example: Deadlock

- Sessions

```
session2> UPDATE t SET f='foo' WHERE id=12345;
ERROR 1213 (40001): Deadlock found when trying to get lock; try restarting transaction
session2> select @@tokudb_last_lock_timeout\G
*************************** 1. row ***************************
@@tokudb_last_lock_timeout: {"mysql_thread_id":2688, "dbname":"./test/t-main",
"requesting_txnid":80, "blocking_txnid":78, "key_left":"ff39300000",  "key_right":"0139300000"}
1 row in set (0.00 sec)

ERROR 1205 (HY000): Lock wait timeout exceeded; try restarting transaction
session1> select @@tokudb_last_lock_timeout\G
*************************** 1. row ***************************
@@tokudb_last_lock_timeout: {"mysql_thread_id":2685, "dbname":"./test/t-main",
"requesting_txnid":78, "blocking_txnid":80, "key_left":"ff31d40000", "key_right":"0131d40000"}
1 row in set (0.00 sec)
```

PERCONA

By example: Deadlock

- Sessions
- Error log

```
2017-03-12T13:45:00.551794Z 2688 [ERROR] TokuDB: lock timeout
{"mysql_thread_id":2688, "dbname":"./test/t-main",
"requesting_txnid":80, "blocking_txnid":78,
"key_left":"ff39300000", "key_right":"0139300000"}
2017-03-12T13:45:00.551827Z 2688 [ERROR] TokuDB:
requesting_thread_id:2688 q:UPDATE t SET f='bar' WHERE id=12345

2017-03-12T13:45:03.863714Z 2685 [ERROR] TokuDB: lock timeout
{"mysql_thread_id":2685, "dbname":"./test/t-main",
"requesting_txnid":78, "blocking_txnid":80,
"key_left":"ff31d40000", "key_right":"0131d40000"}
2017-03-12T13:45:03.863771Z 2685 [ERROR] TokuDB:
requesting_thread_id:2685 q:UPDATE t SET f='foo' WHERE id=54321
```


PERCONA

All TokuDB Transactions

```
information schema> SELECT * FROM TOKUDB_TRX;
+--------+--------------------+----------+
| trx_id | trx_mysql_thread_id | trx_time |
+--------+--------------------+----------+
|    126 |               4135 |       70 |
|    128 |               4141 |       61 |
+--------+--------------------+----------+
2 rows in set (0.00 sec)
```

PERCONA

All TokuDB Locks

```
information schema> SELECT * FROM TOKUDB_LOCKS\G
*************************** 1. row ***************************
               locks_trx_id: 126
      locks_mysql_thread_id: 4135
                locks_dname: ./test/t-main
             locks_key_left: ff39300000
            locks_key_right: 0039300000
          locks_table_schema: test
            locks_table_name: t
locks_table_dictionary_name: main
*************************** 2. row ***************************
               locks_trx_id: 128
      locks_mysql_thread_id: 4141
                locks_dname: ./test/t-main
             locks_key_left: ff31d40000
            locks_key_right: 0031d40000
          locks_table_schema: test
            locks_table_name: t
locks_table_dictionary_name: main
2 rows in set (0.00 sec)
```


PERCONA

TokuDB Lock Waits

```
information schema> SELECT * FROM TOKUDB_LOCK_WAITS\G
*************************** 1. row ***************************
               requesting_trx_id: 146
                  blocking_trx_id: 128
                 lock_waits_dname: ./test/t-main
              lock_waits_key_left: ff31d40000
             lock_waits_key_right: 0131d40000
            lock_waits_start_time: 1489327019006
          lock_waits_table_schema: test
            lock_waits_table_name: t
lock_waits_table_dictionary_name: main
1 row in set (0.00 sec)
```


PERCONA

LOCK IN SHARE MODE

- Not supported
- No error returned
- Clause silently ignored!
- Behaves as regular read lock for SELECT
- Bug #1672532

PERCONA

Lock races

- TokuDB assumes that if some transaction is running lock retry, others do not need to run it

PERCONA

Lock races

- TokuDB assumes that if some transaction is running lock retry, others do not need to run it
- They wait for timeout instead

PERCONA

Lock races

- TokuDB assumes that if some transaction is running lock retry, others do not need to run it
- They wait for timeout instead
- With low number of connections you will sleep when you may do useful job

PERCONA

Lock races

- TokuDB assumes that if some transaction is running lock retry, others do not need to run it
- They wait for timeout instead
- With low number of connections you will sleep when you may do useful job
- Slow performance

PERCONA

Lock races

- TokuDB assumes that if some transaction is running lock retry, others do not need to run it
- They wait for timeout instead
- With low number of connections you will sleep when you may do useful job
- Slow performance
- Pull request to fix it

PERCONA

Performance

Performance

Query Performance

PERCONA

Clustering Secondary Indexes

```
create clustering index birth_date on employees(birth_date);
alter table employees add clustering index (birth_date);
```

PERCONA

Clustering Secondary Indexes

- Contains copy of data

PERCONA

Clustering Secondary Indexes

- Contains copy of data
- Practically ordered copy of the table

PERCONA

Clustering Secondary Indexes

- Contains copy of data
- Practically ordered copy of the table
- Covered index for any kind of query
  - "Regular" index

```
session1> create index birth_date on employees(birth_date);
Query OK, 0 rows affected (4.99 sec)
Records: 0  Duplicates: 0  Warnings: 0

session1> select first_name, last_name, birth_date from employees
       -> where birth_date > '1960-01-01' order by birth_date desc;
...
117075 rows in set (1.27 sec)
```


PERCONA

Clustering Secondary Indexes

- Contains copy of data
- Practically ordered copy of the table
- Covered index for any kind of query
  - Clustering index

```
session1> create clustering index birth_date on employees(birth_date);
Query OK, 0 rows affected (5.96 sec)
Records: 0  Duplicates: 0  Warnings: 0

session1> select first_name, last_name, birth_date from employees
       -> where birth_date > '1960-01-01';
...
117075 rows in set (0.69 sec)
```

PERCONA

Clustering Secondary Indexes

- Contains copy of data
- Practically ordered copy of the table
- Covered index for any kind of query
- Cannot be unique

PERCONA

Index Statistics

- Prior to 5.6.27-76.0
  - ANALYZE TABLE in foreground

PERCONA

Index Statistics

- Prior to 5.6.27-76.0
- Variable tokudb_analyze_in_background
  - ANALYZE TABLE in background
  - Immediate return

PERCONA

Index Statistics

- Prior to 5.6.27-76.0
- Variable tokudb_analyze_in_background
- Controlled by variable tokudb_analyze_mode
  - TOKUDB_ANALYZE_CANCEL
  - TOKUDB_ANALYZE_STANDARD
  - TOKUDB_ANALYZE_RECOUNT_ROWS

PERCONA

Index Statistics

- Prior to 5.6.27-76.0
- Variable tokudb_analyze_in_background
- Controlled by variable tokudb_analyze_mode
- How ANALYZE operation is aggressive
  - tokudb_analyze_throttle: maximum number of keys to visit per second
  - tokudb_analyze_time: number of seconds to spend on each index when calculating cardinality

PERCONA

Index Statistics

- Prior to 5.6.27-76.0
- Variable tokudb_analyze_in_background
- Controlled by variable tokudb_analyze_mode
- How ANALYZE operation is aggressive
- Variable tokudb_auto_analyze
  - Fully automatic ANALYZE
  - Similar to innodb_stats_auto_recalc
  - Enabled in thread context which triggered analysis

PERCONA

Index Statistics

- Prior to 5.6.27-76.0
- Variable tokudb_analyze_in_background
- Controlled by variable tokudb_analyze_mode
- How ANALYZE operation is aggressive
- Variable tokudb_auto_analyze
- Variable tokudb_cardinality_scale_percent
  - Tells server how index is unique
  - Hardcoded in InnoDB: 50%

PERCONA

Index Condition Pushdown

- Not all ICP conditions are implemented

PERCONA

Index Condition Pushdown

- Not all ICP conditions are implemented
- No mechanism to pass information about ICP usage back to the server
  - You will not see "Using index condition" for TokuDB even if optimization used

PERCONA

Index Condition Pushdown

- Not all ICP conditions are implemented
- No mechanism to pass information about ICP usage back to the server
  - You will not see "Using index condition" for TokuDB even if optimization used
- Descending range scans were pain before 5.6.35/5.7.17
  - `SELECT id FROM t1 WHERE id BETWEEN 5 AND 5000 ORDER BY id DESC`
  - Bug #1672871

PERCONA

OPTIMIZE TABLE

- Two block sets
  - Crash safe

PERCONA

OPTIMIZE TABLE

- Two block sets
  - Crash safe
- OPTIMIZE TABLE changes nodes

PERCONA

OPTIMIZE TABLE

- Two block sets
  - Crash safe
- OPTIMIZE TABLE changes nodes
- Nodes are flushed into new block set

PERCONA

OPTIMIZE TABLE

- Two block sets
  - Crash safe
- OPTIMIZE TABLE changes nodes
- Nodes are flushed into new block set
- Old block set still exists

PERCONA

OPTIMIZE TABLE

- Two block sets
  - Crash safe
- OPTIMIZE TABLE changes nodes
- Nodes are flushed into new block set
- Old block set still exists
- Size of index doubles

PERCONA

OPTIMIZE TABLE

- Two block sets
  - Crash safe
- OPTIMIZE TABLE changes nodes
- Nodes are flushed into new block set
- Old block set still exists
- Size of index doubles
- These bytes will be re-used
  - Kind of pre-allocation

PERCONA

Online CREATE INDEX

- Special syntax

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

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


PERCONA

Online CREATE INDEX

- Special syntax
- ALTER implies offline index creation

PERCONA

Online CREATE INDEX

- Special syntax
- ALTER implies offline index creation
- Watch progress in SHOW PROCESSLIST

```
session2> show processlist\G
*************************** 1. row ***************************
           Id: 424
         User: root
         Host: localhost:40422
           db: employees
      Command: Query
         Time: 7
        State: Adding of indexes about 98.4% done
         Info: create index title on titles(title)
    Rows_sent: 0
Rows_examined: 0
```


PERCONA

Query progress

- Read progress

```
session2> show processlist\G
*************************** 1. row ***************************
           Id: 424
         User: root
         Host: localhost:40422
           db: employees
      Command: Query
         Time: 1
        State: Queried about 300000 rows
         Info: select title, count(*) from titles group by title
    Rows_sent: 0
Rows_examined: 0
```

PERCONA

Query progress

- Read progress
- Write progress

```
session2> show processlist\G
*************************** 1. row ***************************
           Id: 424
         User: root
         Host: localhost:40422
           db: employees
      Command: Query
         Time: 6
        State: Queried about 443308 rows, Updated about 53000 rows
         Info: update titles set title='Senior Cat' where title='Senior Engineer'
    Rows_sent: 0
Rows_examined: 496933
```

PERCONA

Performance

High Concurrency Performance

PERCONA

Main Factors which Affect Performance

- Checkpointing

PERCONA

Main Factors which Affect Performance

- Checkpointing
- Read heavy workloads
  - OS cache with high compression levels may work well
  - TokuDB is write optimized!

PERCONA

Main Factors which Affect Performance

- Checkpointing
- Read heavy workloads
- Fractal Tree map file fragmentation
  - Block allocator algorithm
  - Improved in newer versions
    - Old versions: linear search
    - New versions: MHS algorithm
    - Details

PERCONA

Main Factors which Affect Performance

- Checkpointing
- Read heavy workloads
- Fractal Tree map file fragmentation
- Concurrent queries on single table

PERCONA

TokuDB Sharp Checkpointing

- Flushing down dirty pages in background

PERCONA

TokuDB Sharp Checkpointing

- Flushing down dirty pages in background
- Fuzzy checkpointing (as in InnoDB)
  - Flushes dirty pages when server is idle
  - Or if innodb_max_dirty_pages_pct reached

PERCONA

TokuDB Sharp Checkpointing

- Flushing down dirty pages in background
- Fuzzy checkpointing (as in InnoDB)
- Sharp checkpoining (as in TokuDB)
  - It will checkpoint at some time period no matter what the current workload is
  - May lead to visible stalls in user activity
  - Variables tokudb_checkpointing_period and tokudb_cache_size control how often checkpointing will happen

PERCONA

Read Heavy Workloads

- Fractal Tree design
  - Fractal Tree buffers modifications in intermediate locations
  - These buffered modifications should be eventually flushed into index files
  - While they are not flushed read operations have to search through all not applied stages

PERCONA

Read Heavy Workloads

- Fractal Tree design
- Write locks have higher priority than read locks
  - Trx 1 blocks a row with read lock

PERCONA

Read Heavy Workloads

- Fractal Tree design
- Write locks have higher priority than read locks
  - Trx 1 blocks a row with read lock
  - Trx 2 requests write lock on this row

PERCONA

Read Heavy Workloads

- Fractal Tree design
- Write locks have higher priority than read locks
  - Trx 1 blocks a row with read lock
  - Trx 2 requests write lock on this row
  - Trx 3 requests read lock on the row

PERCONA

Read Heavy Workloads

- Fractal Tree design
- Write locks have higher priority than read locks
  - Trx 1 blocks a row with read lock
  - Trx 2 requests write lock on this row
  - Trx 3 requests read lock on the row
  - InnoDB: Trx 3 receives access immediately

PERCONA

Read Heavy Workloads

- Fractal Tree design
- Write locks have higher priority than read locks
  - Trx 1 blocks a row with read lock
  - Trx 2 requests write lock on this row
  - Trx 3 requests read lock on the row
  - TokuDB: Trx 3 will wait when Trx 2 releases lock

PERCONA

Instruments

Tables in Information Schema

- TokuDB_trx

```
session2> select * from TokuDB_trx\G
*************************** 1. row ***************************
            trx_id: 4022
trx_mysql_thread_id: 424
          trx_time: 7
1 row in set (0.00 sec)
```

Tables in Information Schema

- TokuDB_locks

```
session2> select * from TokuDB_locks \G
*************************** 1. row ***************************
              locks_trx_id: 4022
      locks_mysql_thread_id: 424
               locks_dname: ./employees/titles-main
            locks_key_left: 00a16304000a53656e696f722043617479840f
           locks_key_right: 00a16304000a53656e696f722043617479840f
      locks_table_schema: employees
        locks_table_name: titles
locks_table_dictionary_name: main
...
```

Tables in Information Schema

- TokuDB_lock_waits

```
mysql> select * from information_schema.tokudb_lock_waits\G
*************************** 1. row ***************************
               requesting_trx_id: 4115
                  blocking_trx_id: 4022
                 lock_waits_dname: ./employees/titles-main
              lock_waits_key_left: -infinity
             lock_waits_key_right: +infinity
           lock_waits_start_time: 1489526456023
         lock_waits_table_schema: employees
           lock_waits_table_name: titles
lock_waits_table_dictionary_name: main
1 row in set (0.00 sec)
```


PERCONA

Tables in Information Schema

- TokuDB_background_job_status

```
session1> select * from information_schema.TokuDB_background_job_status\G
*************************** 1. row ***************************
            id: 35
 database_name: employees
    table_name: titles
      job_type: TOKUDB_ANALYZE_MODE_RECOUNT_ROWS
    job_params: TOKUDB_ANALYZE_THROTTLE=0;
     scheduler: USER
scheduled_time: 2017-03-15 00:37:10
  started_time: 2017-03-15 00:37:10
        status: recount_rows employees.titles counted 230230 rows and 0 deleted in 0 seconds.
1 row in set (0.00 sec)
```

PERCONA

Tables in Information Schema

- TokuDB_file_map

```
session1> show create table departments\G
*************************** 1. row ***************************
       Table: departments
Create Table: CREATE TABLE `departments` (
  `dept_no` char(4) NOT NULL,
  `dept_name` varchar(40) NOT NULL,
  PRIMARY KEY (`dept_no`),
  UNIQUE KEY `dept_name` (`dept_name`)
) ENGINE=TokuDB DEFAULT CHARSET=latin1
1 row in set (0.00 sec)
```


PERCONA

Tables in Information Schema

- TokuDB_file_map

```
mysql> select * from TokuDB_file_map\G
*************************** 1. row ***************************
      dictionary_name: ./employees/departments-key-dept_name
   internal_file_name: ./employees/departments_key_dept_name_5ca_1_1d_B_1.tokudb
         table_schema: employees
           table_name: departments
table_dictionary_name: key-dept_name
*************************** 2. row ***************************
      dictionary_name: ./employees/departments-main
   internal_file_name: ./employees/departments_main_5ca_1_1d_B_0.tokudb
                  ...
table_dictionary_name: main
*************************** 3. row ***************************
                  ...
table_dictionary_name: status
```

PERCONA

Tables in Information Schema

- TokuDB_fractal_tree_block_map

```
mysql> select * from TokuDB_fractal_tree_block_map where dictionary_name
    -> like './employees/departments%'\G
*************************** 1. row ***************************
      dictionary_name: ./employees/departments-key-dept_name
   internal_file_name: ./employees/departments_key_dept_name_5ca_1_1d_B_1.tokudb
     checkpoint_count: 1
            blocknum: 0
              offset: NULL
                size: 0
        table_schema: employees
          table_name: departments
table_dictionary_name: key-dept_name
```

PERCONA

Tables in Information Schema

- TokuDB_fractal_tree_info

```
mysql> select * from TokuDB_fractal_tree_info where dictionary_name
    -> like './employees/departments%'\G
*************************** 1. row ***************************
        dictionary_name: ./employees/departments-key-dept_name
     internal_file_name: ./employees/departments_key_dept_name_5ca_1_1d_B_1.tokudb
bt_num_blocks_allocated: 4
   bt_num_blocks_in_use: 4
      bt_size_allocated: 12360
         bt_size_in_use: 584
           table_schema: employees
             table_name: departments
  table_dictionary_name: key-dept_name
```

PERCONA

SHOW ENGINE TOKUDB STATUS

- Internal information about TokuDB engine

```
mysql> SHOW ENGINE TOKUDB STATUS;
+--------+-----------------------------------------------------------------+-------------------------+
| Type   | Name                                                            | Status                  |
+--------+-----------------------------------------------------------------+-------------------------+
| TokuDB | disk free space                                                 | more than 10 percent... |
| TokuDB | time of environment creation                                    | Sun Mar 12 15:58:05 2017 |
| TokuDB | time of engine startup                                          | Tue Mar 14 22:44:31 2017 |
| TokuDB | time now                                                        | Wed Mar 15 01:05:31 2017 |
| TokuDB | db opens                                                        | 891                     |
| TokuDB | db closes                                                       | 881                     |
| TokuDB | num open dbs now                                                | 10                      |
| TokuDB | max open dbs                                                    | 23                      |
| TokuDB | period, in ms, that recovery log is automatically fsynced       | 0                       |
| TokuDB | dictionary inserts                                              | 3522300                 |
...
```


PERCONA

SHOW ENGINE TOKUDB STATUS

- Internal information about TokuDB engine

```
| TokuDB | le: max committed xr         | 2    |
...
| TokuDB | checkpoint: period           | 60   |
...
| TokuDB | cachetable: miss             | 23   |
...
| TokuDB | locktree: memory size        | 0    |
...
| TokuDB | ft: dictionary updates       | 0    |
...
| TokuDB | hot: operations ever started | 0    |
...
| TokuDB | txn: begin                   | 4436 |
| TokuDB | txn: begin read only         | 2    |
| TokuDB | txn: successful commits      | 4419 |
...
```

PERCONA

SHOW ENGINE TOKUDB STATUS

- Internal information about TokuDB engine

```
| TokuDB | logger: next LSN                                    | 4620325  |
...
| TokuDB | indexer: number of indexers successfully created    | 1        |
...
| TokuDB | loader: number of loaders successfully created      | 8        |
...
| TokuDB | memory: number of malloc operations                 | 0        |
...
| TokuDB | filesystem: ENOSPC redzone state                    | 0        |
| TokuDB | filesystem: threads currently blocked by full disk  | 0        |
...
| TokuDB | context: tree traversals blocked by a full fetch    | 0        |
...
| TokuDB | handlerton: primary key bytes inserted              | 67840548 |
+--------+-----------------------------------------------------+----------+
322 rows in set (0.05 sec)
```


PERCONA

Status Variables

- Inner workings of TokuDB storage engine
- Headers, similar to ENGINE TOKUDB STATUS

PERCONA

Server Variables

- Help to tune TokuDB

PERCONA

Server Variables

- Help to tune TokuDB
- Several are really status variables
  - tokudb_backup_last_error[_string]
  - tokudb_last_lock_timeout

PERCONA

Server Variables

- Help to tune TokuDB
- Several are really status variables
  - tokudb_backup_last_error[_string]
  - tokudb_last_lock_timeout
- Can trigger operations
  - tokudb_backup_dir

PERCONA

Summary

Summary

- TokuDB is fully functional engine
- Reach lock diagnostic
- Reach progress diagnostic
- Engine runtime information
  - SHOW ENGINE TOKUDB STATUS
  - Status variables
- Write optimized

PERCONA

More information

- Documentation in the user manual
- TokuDB posts at Percona Blog
- Old Tokutek posts in Percona Blog

PERCONA

Time for questions

???

PERCONA

Thank you!

http://www.slideshare.net/SvetaSmirnova

https://twitter.com/svetsmirnova

PERCONA

DATABASE PERFORMANCE
MATTERS