TokuDB vs RocksDB

What to choose between two write-optimized DB engines supported by Percona

George O. Lorch III
Vlad Lesin

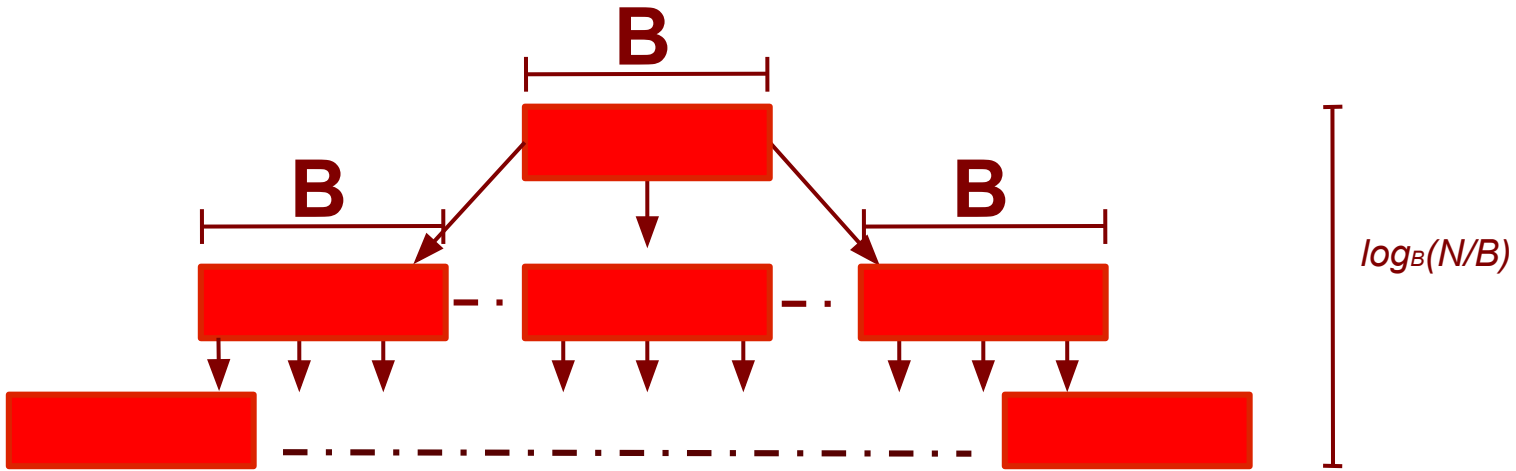
What to compare?

- Amplification
  - Write amplification
  - Read amplification
  - Space amplification
- Features set

How to compare?

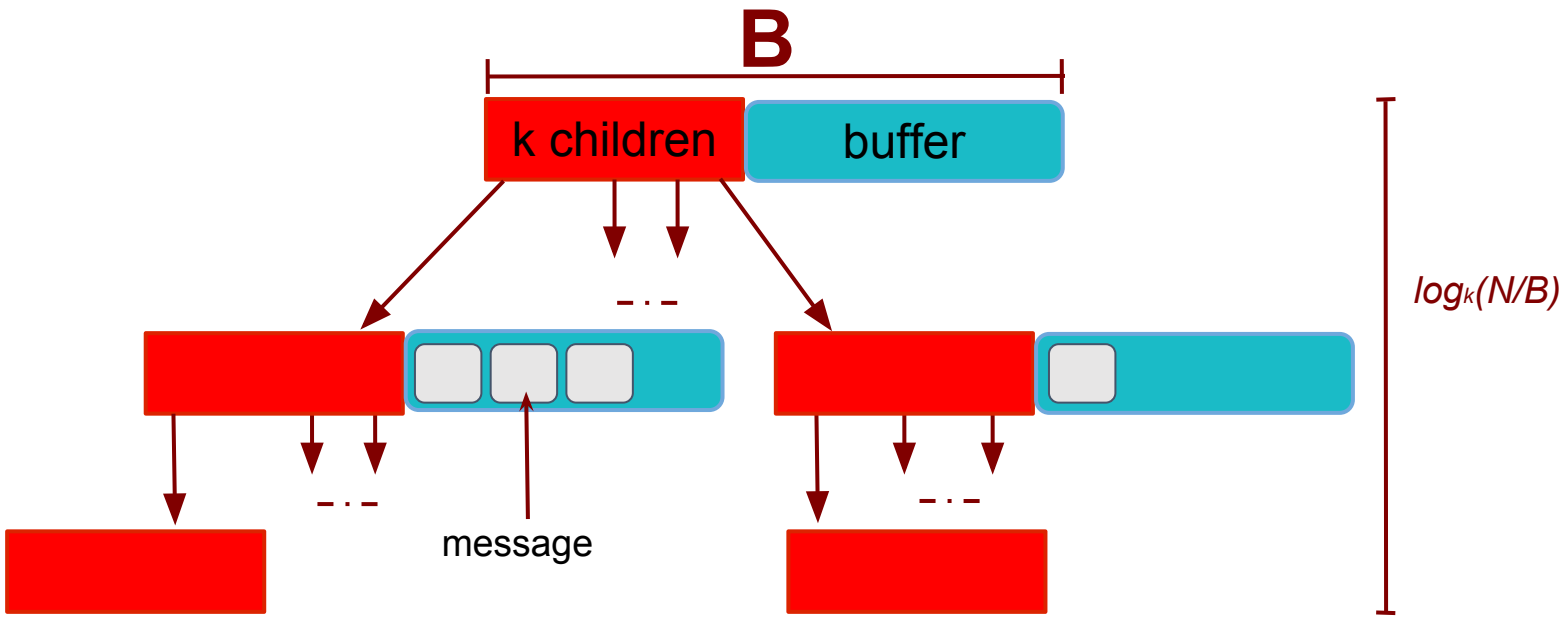
- Theoretical review
  - Fractal trees vs LSM
- Implementation
  - What and how is persistently stored?
  - What and how is cached?
  - Read/write/space tradeoff options
- Benchmarks

Fractal tree: B-tree



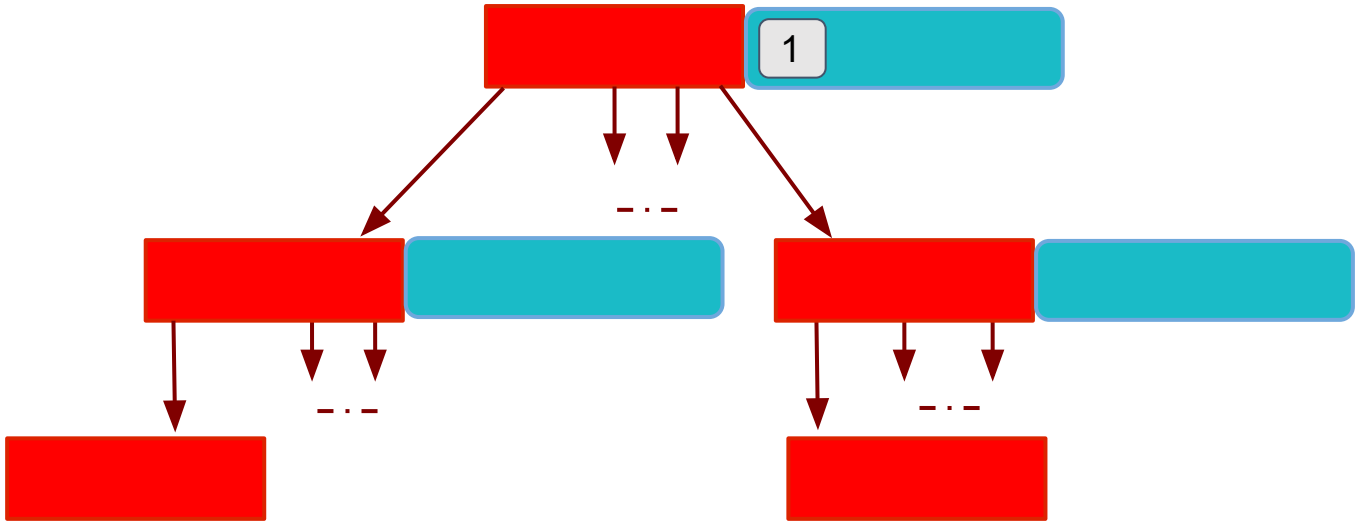
PERCONA

Fractal tree: B-tree with node buffers



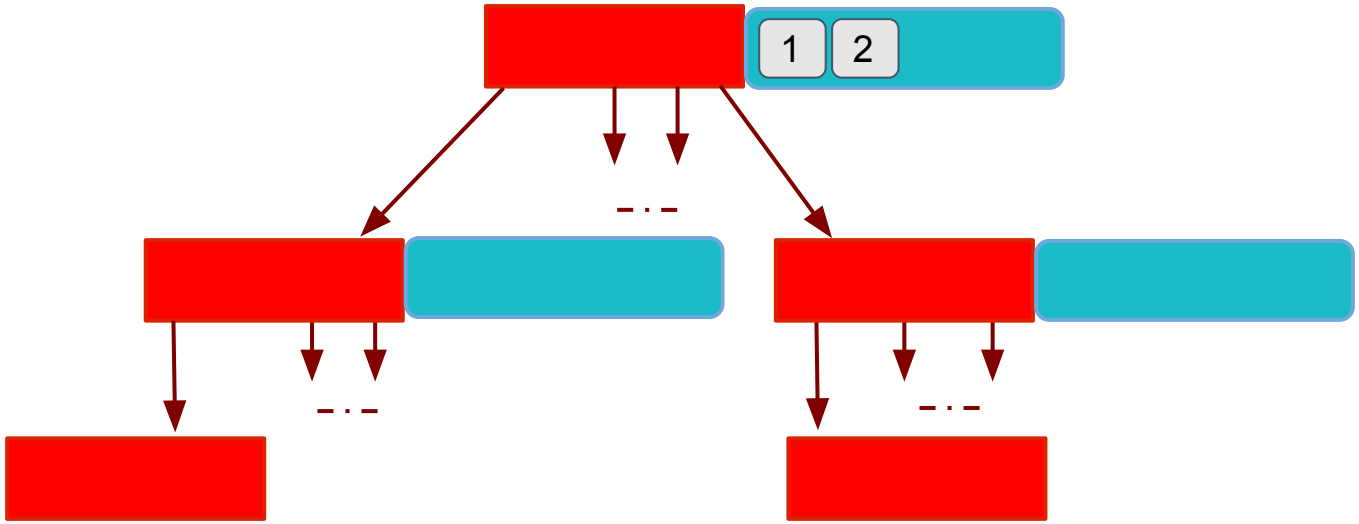
PERCONA

Fractal tree: messages pushdown



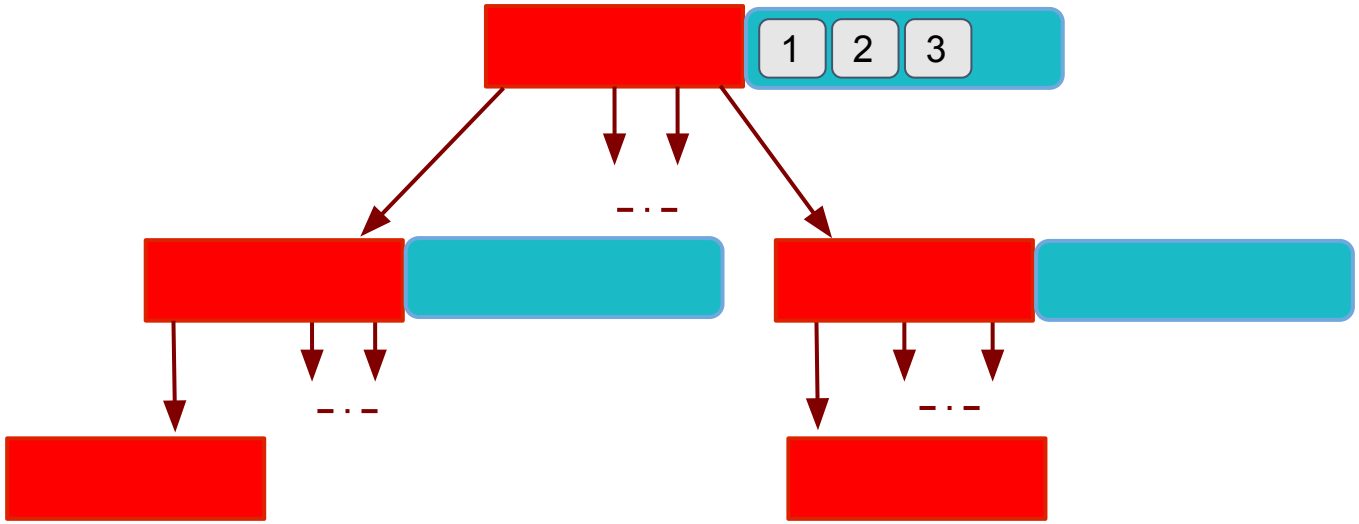
PERCONA

Fractal tree: messages pushdown



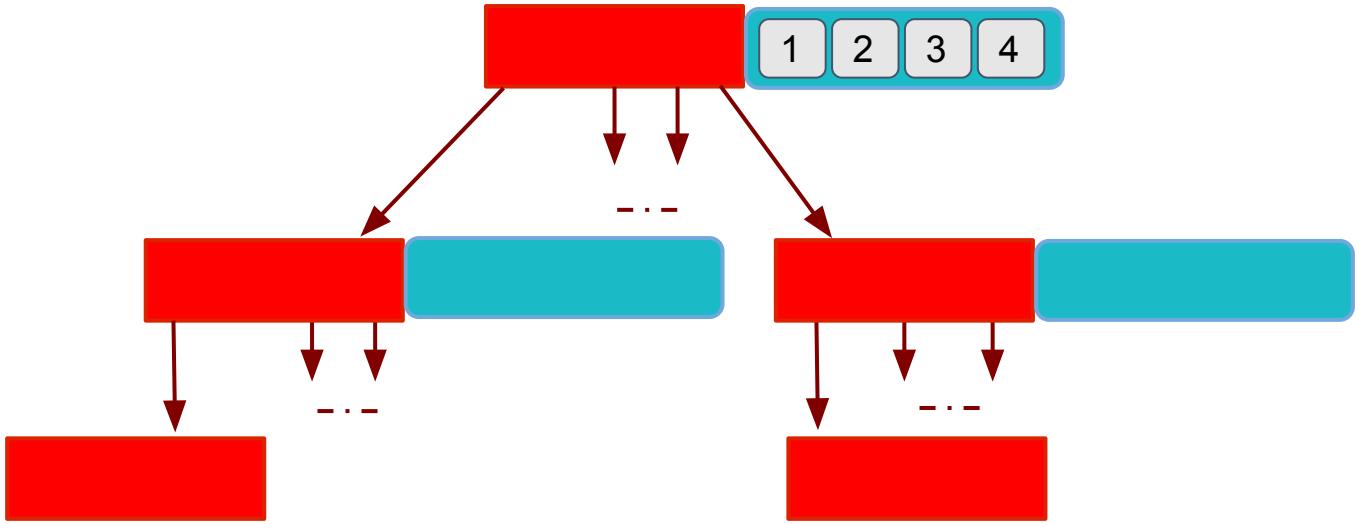
PERCONA

Fractal tree: messages pushdown



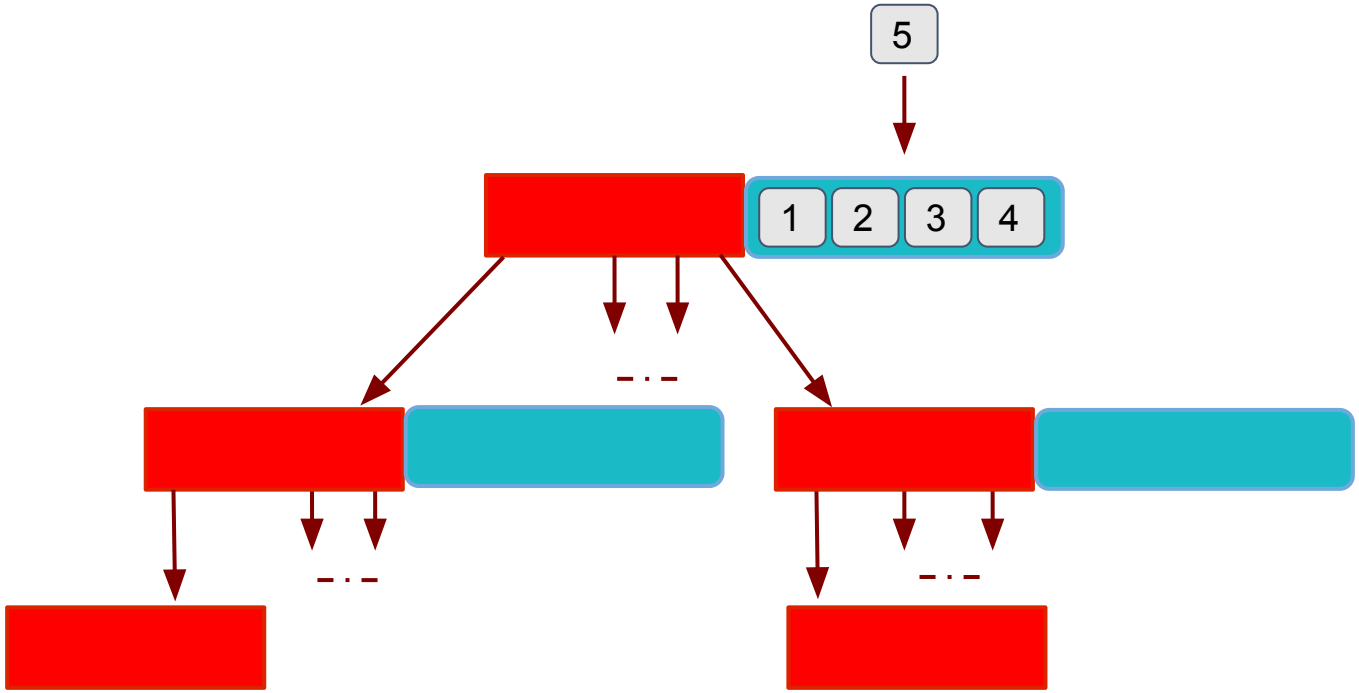
PERCONA

Fractal tree: messages pushdown



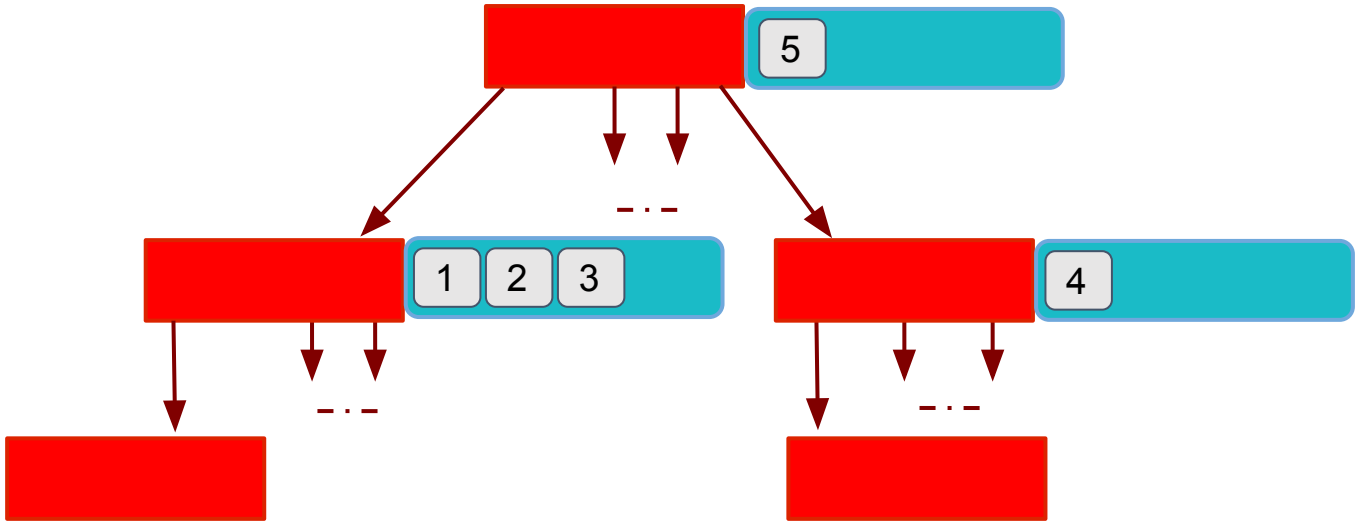
PERCONA

Fractal tree: messages pushdown



PERCONA

Fractal tree: messages pushdown



PERCONA

Fractal tree: amplification for both point and range

- Write amplification
  - Worst case: $O(k*\log_k(N/B))$
  - Skewed writes: $O(\log_k(N/B))$
- Read amplification
  - Cold cache: $O(\log_k(N/B))$
  - Warm cache: 1

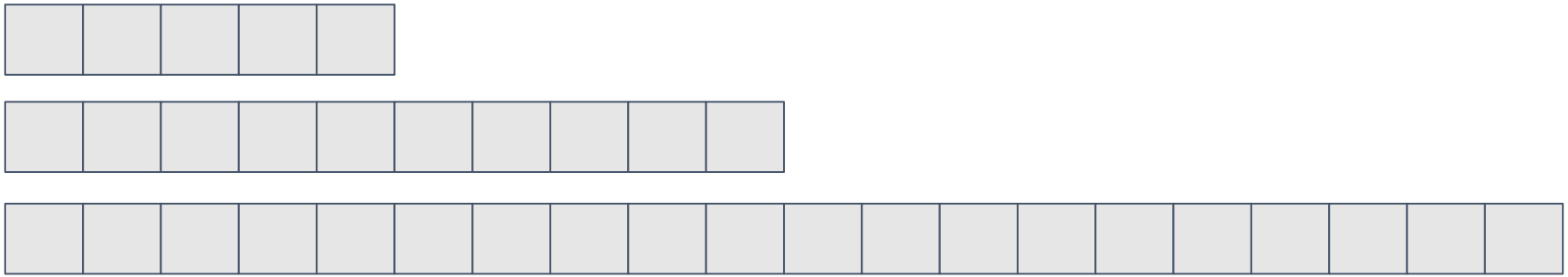
where k - fanout, B - block size, N - data size.

LSM: Log Structured Merge tree

- A collection of sorted runs of data
- Each run contains kv-pairs with keys sorted
- A run can be managed as a single file or a collection of smaller files
- Binary search

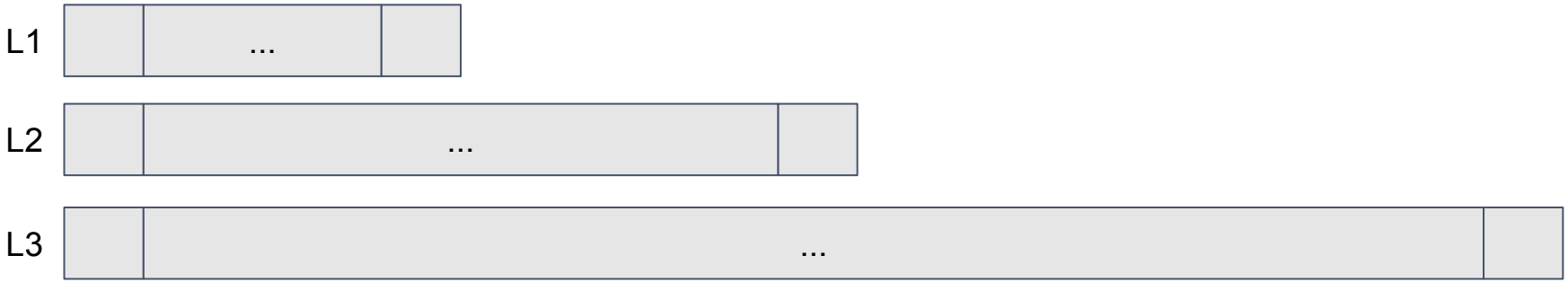
LSM: example

- Consist of collection of runs
- Many versions of a row can exist in different runs
- Binary search in each run
- Compaction to maintain good read performance



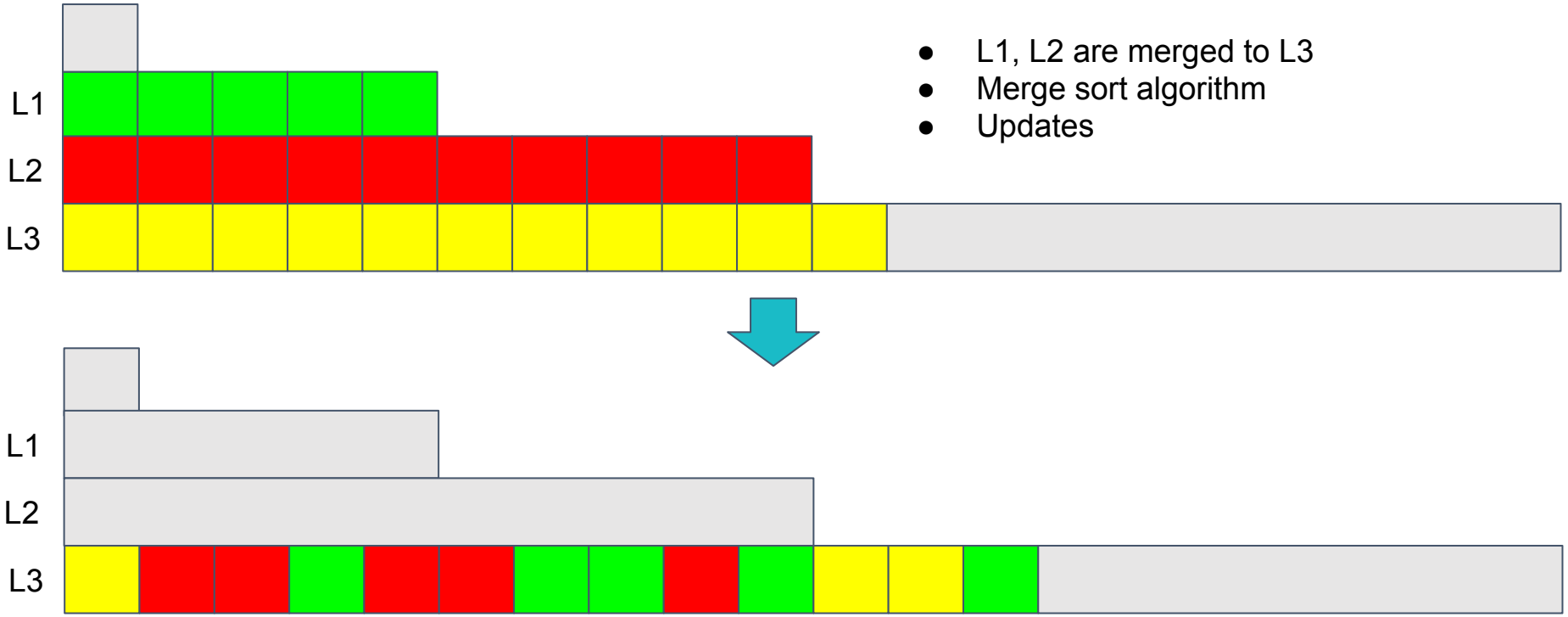
LSM: leveled

- Each level is limited by size
- For example if 10^(i-1)< sizeof(Li) <= 10^i then
  - L1 is between 1MB and 10MB
  - L2 is between 10MB and 100MB
  - L3 is between 100MB and 1Gb
  - ...
- When the size of certain level reaches it’s limit the level is merged to the next level



PERCONA

LSM: compaction example

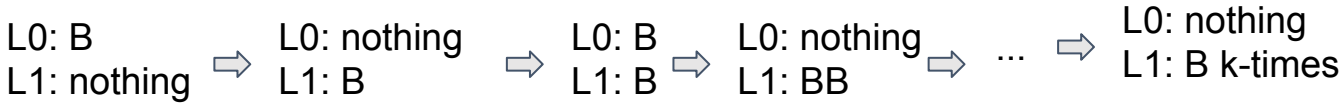


- L1, L2 are merged to L3
- Merge sort algorithm
- Updates

PERCONA

LSM: write amplification

- $\Theta(\log_k(N/B))$ levels where k - growth factor, N - data size, B - block size.
- On average each block is remerged back at the same level k/2 times?



The first B bytes is remerged back in L1 k-1 times, the second B bytes are remerged back k-1 times, and so on, the last B bytes are remerged back 0 times. The average number of remerged bytes is sum(number of remerges for kB bytes)/k. The sum is arithmetic progression and can be counted as (k^2)/2, so the average number of remerges is k/2.

- The total write amplification is $\Theta(k\log_k(N/B))$

LSM: read amplification

- For range queries and cold cache summarizing binary search read amplification for all runs: O(((log(N/B))^2)/log(k))
- For range queries and warm cache the first several runs are cached so range read will cost several IOs
- For point queries and bloom filters 1 IO in most cases.

FT vs LSM: resume

Asymptotically:

- FT and LSM are the same for write amplification
- FT is better than LSM for range reads on cold cache, but the same on warm cache
- FT is the same as LSM for point reads: one or several IO’s per read

TokuDB vs MyRocks: implementation

- How data is stored on disk?
- What and how data is cached?

TokuDB vs MyRocks: storage files

TokuDB

- Log files + block files
- The only FT in each file
- File per index

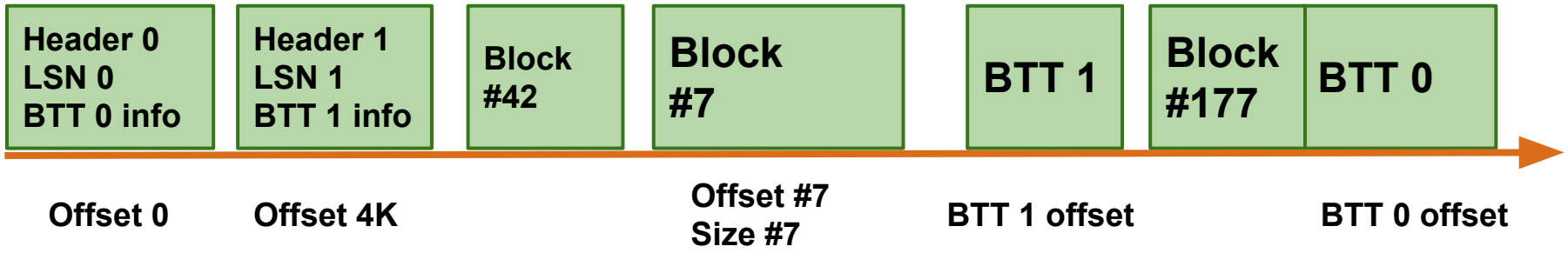
MyRocks

- WAL files + SST files
- All tables and indexes are in the same space(s)
- ColumnFamilies

TokuDB: block file

- Each block is a tree node
- The blocks can have different size
- Two sets of blocks, the sets can intersect

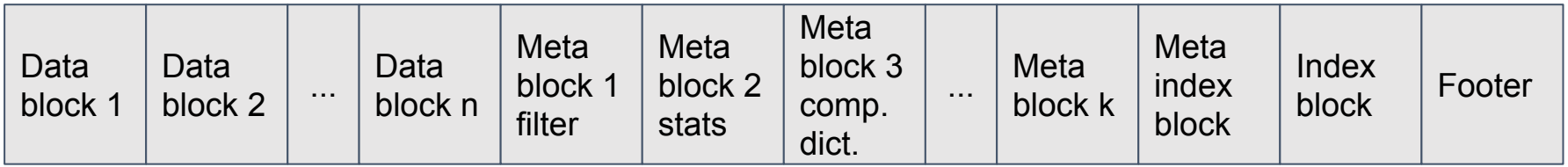
- There can be used and free blocks
- The blocks can be compressed
- Blocks translation table
- Fragmentation



PERCONA

MyRocks: block-based sst file format

- kv-pairs are ordered and partitioned in data blocks
- Meta-index contains references to meta blocks
- Index block contains one entry per data block which contains it’s the last key
- Bloom filter per block or per file
- Compression dictionary for the whole sst file



PERCONA

TokuDB vs MyRocks: persistent storage

TokuDB

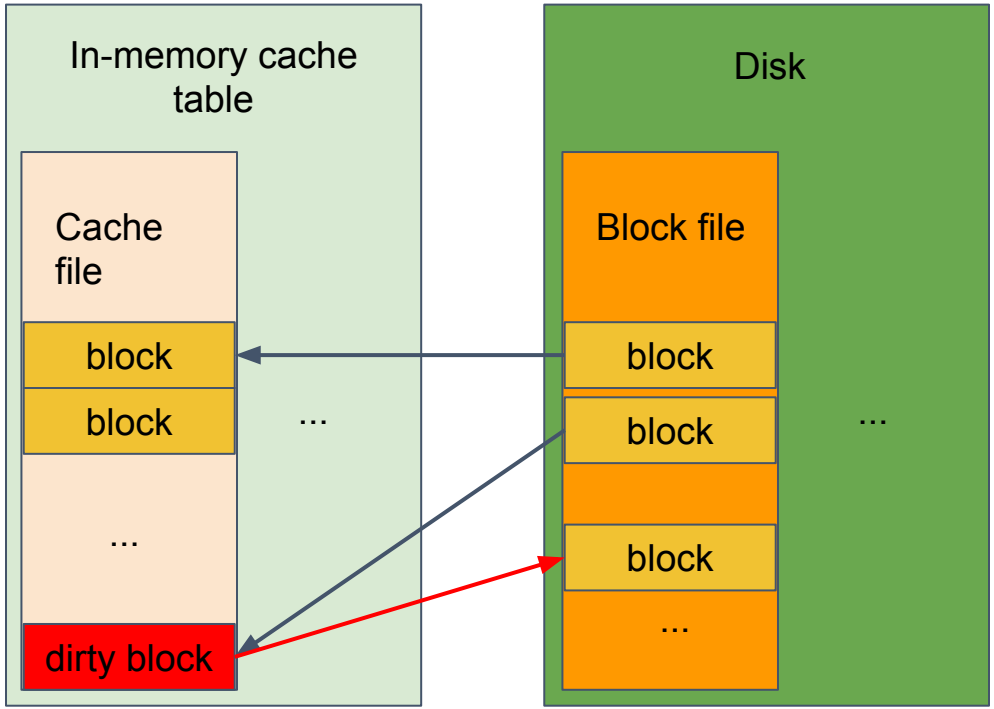
- Two sets of blocks, the sets can be intersected, but on heavy random updates the intersection is small, the free blocks are reallocated
- One file per index(drop index is just file removing)
- Big blocks compressed

RocksDB

- There are no gaps like in TokuDB, but on heavy random updates there can be several values for the same key in different levels (which are merge on compaction)
- All indexes in the same namespace(s)
- Data blocks are compressed, compressed dictionary is common

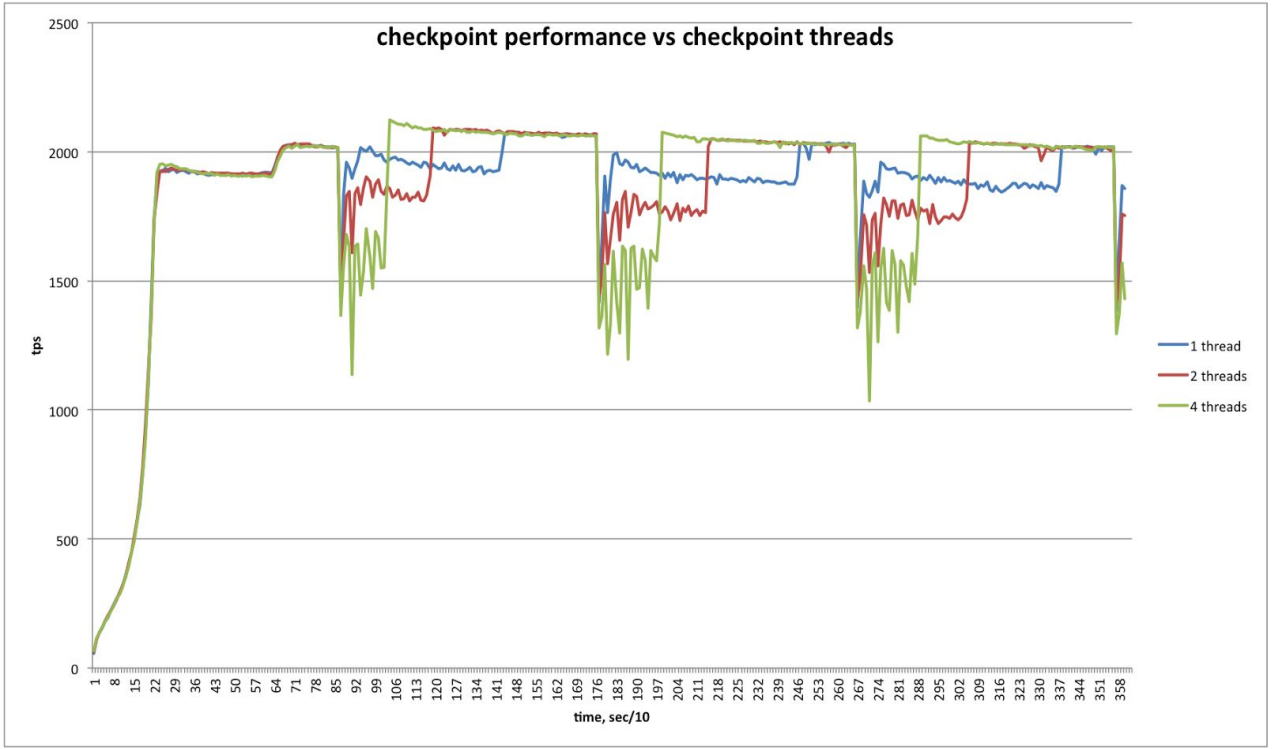
TokuDB: caching

- Blocks are cached in cache table
- Leaf nodes can be read partially
- All modifications are in memory
- Dirty pages are flushed during checkpoint or eviction
- Checkpoint is periodical
  - mark all nodes as “pending”
  - If client modifies “pending” node it’s cloned
- Checkpoint can lead to extra io and memory usage
- Clock eviction algorithm



PERCONA

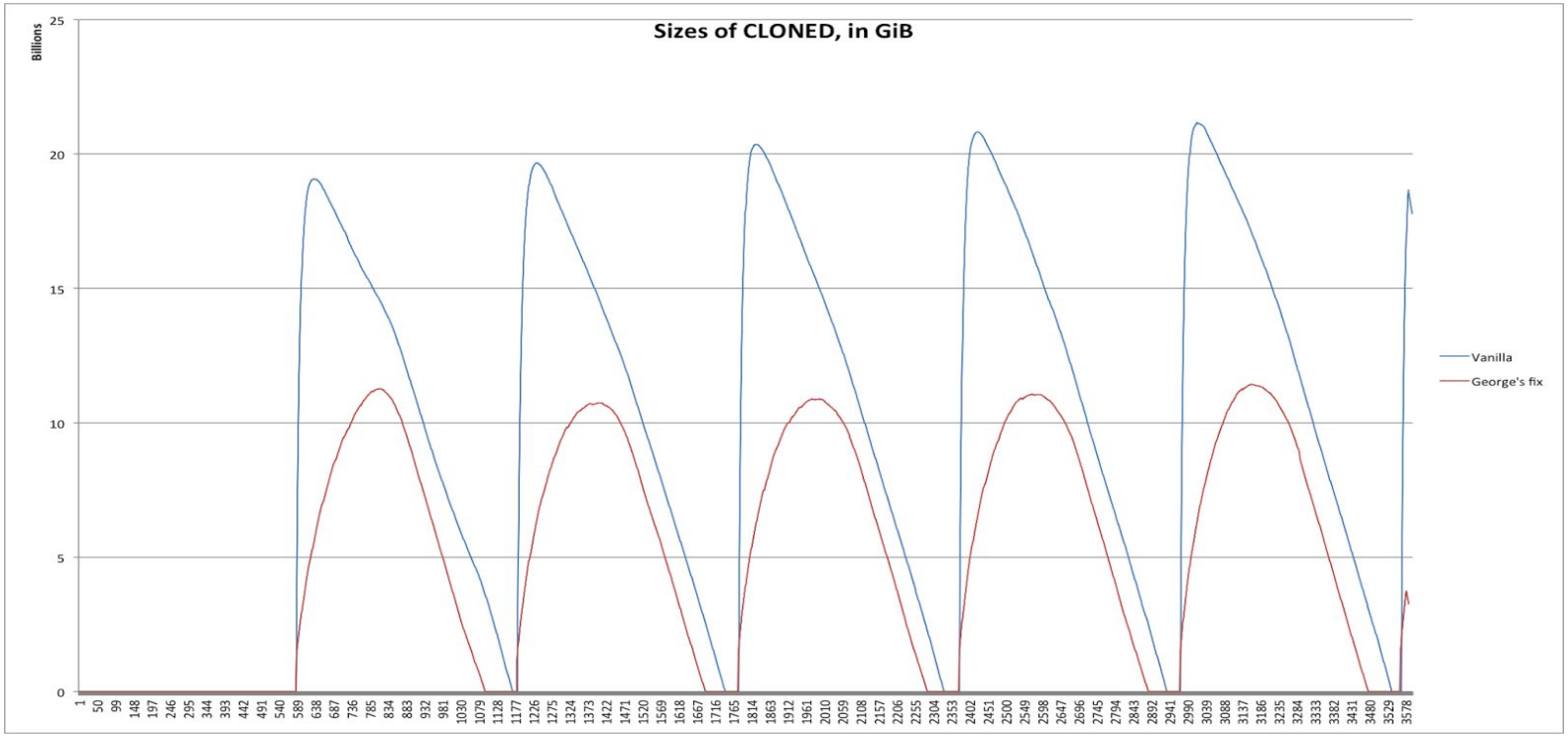
TokuDB: checkpoint performance



PERCONA

TokuDB: checkpoints and clone storm effect

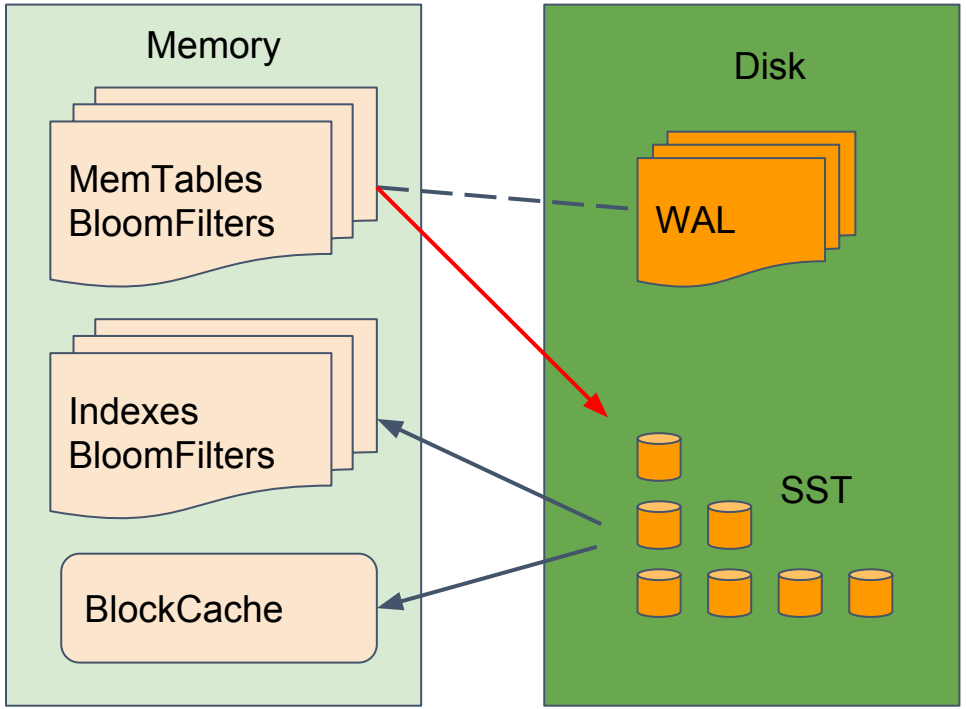
TokuDB: checkpoints and clone storm effect



PERCONA

MyRocks: caching

- Block Cache
  - Sharded LRU
  - Clock
- Indexes and block file bloom filters are caches separately by default, they don't count in memory allocated for BlockCache and can be big memory contributors.
- MemTables can be considered as write buffers, the more space for MemTables, the less write amplification, several kinds of memtable are supported



PERCONA

TokuDB vs MyRocks: benchmarking

- Different data structures to store data persistently
- Both engines have settings for read/write/space tradeoff
- Mysql infrastructure influence
- Different set of features

TokuDB vs MyRocks: benchmarking

Hard to compare because each engine has settings for read-write tradeoff tuning

- TokuDB: block size, fanout, etc...
- MyRocks: blocks size, base level size, etc…

MySQL Transaction Coordinator issues

- > 1 transactional storage engine ***installed*** requires a TC for every transaction.
- Binlog is considered a transactional storage engine when enabled.
- MySQL provides two functioning coordinators, TC_BINLOG and TC_MMAP.
- When binary logging is enabled TC_BINLOG is always used, otherwise TC_MMAP is used when > 1 transactional engine installed.
- TC_MMAP is not optimized nor has as good of a group commit implementation as TC_BINLOG. When TC_MMAP is used, dramatic increase in fsyncs.
- Considering options to best improve this in Percona Server:
  - Add session variable to tell if TC is needed for next transaction and return SQL error if transaction tries to engage > 1 transactional storage engine.
  - Remove TC_MMAP and enhance TC_BINLOG to allow use without actual binary logs, i.e. keep TC logic but disable binlog logic.

TokuDB vs MyRocks: benchmarking

Table:

```
CREATE TABLE sbtest1 (
        id INTEGER NOT NULL,
        k INTEGER DEFAULT '0' NOT NULL,
        c CHAR(120) DEFAULT '' NOT NULL,
        pad CHAR(60) DEFAULT '' NOT NULL,
        PRIMARY KEY (id)) ENGINE = ...
```

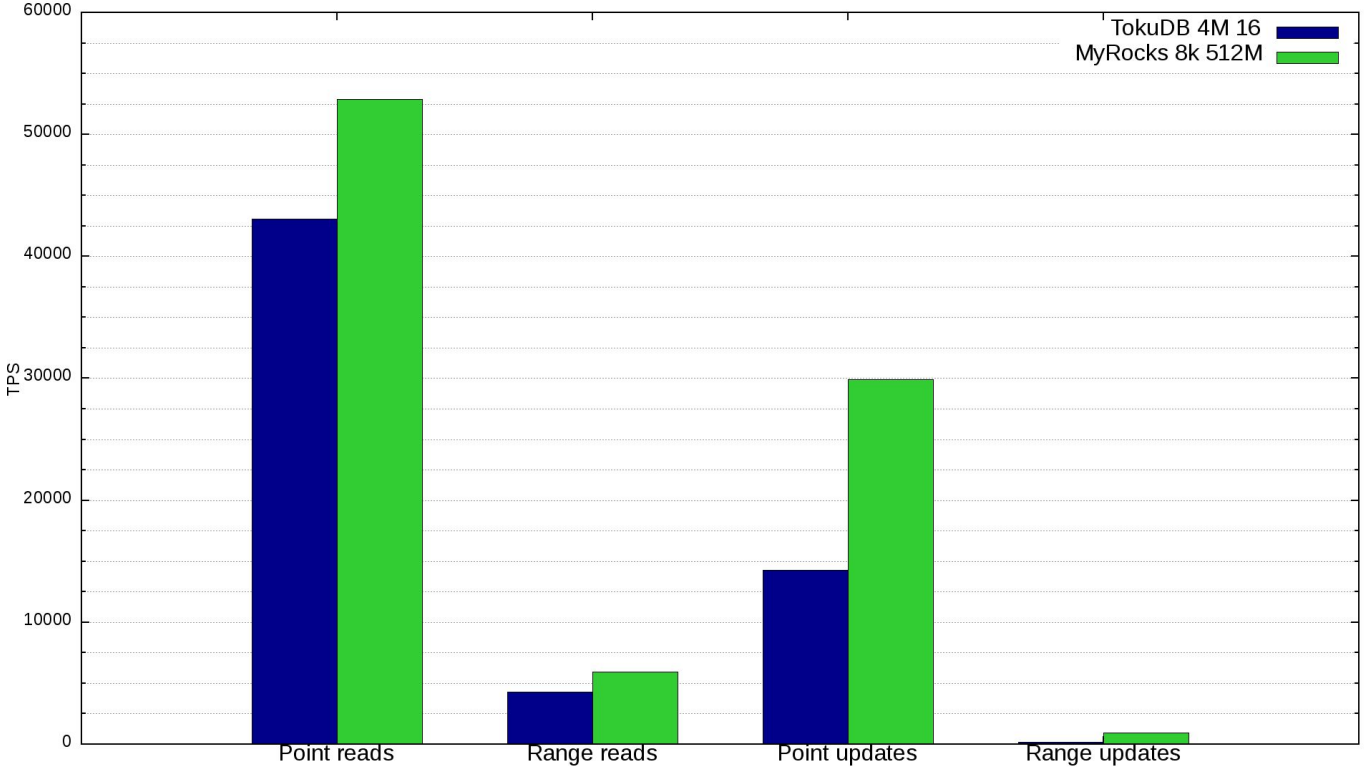
TokuDB vs MyRocks: benchmarking

- Point reads: SELECT id, k, c FROM sbtest1 WHERE id IN (?) - 10 random points
- Range reads: SELECT count(id) FROM sbtest1 id BETWEEN ? AND ? OR…, number of ranges 10, interval - 300
- Point updates: UPDATE sbtest1 SET c=? WHERE id=?
- Range updates: UPDATE sbtest1 SET c=? WHERE id BETWEEN ? AND ?, interval=100
- recovery logs are not synchronized on each commit

TokuDB vs MyRocks: benchmarking, first case

- NVME, ~400G table size, 10G - DB cache, 10G - fs cache
- 48 HW threads, 40 sysbench threads
- TokuDB: block size 4M, fanout 16
- MyRocks: block size 8k, first level size 512M
    - 5 levels for 400G table size

TokuDB vs MyRocks: TPS first case

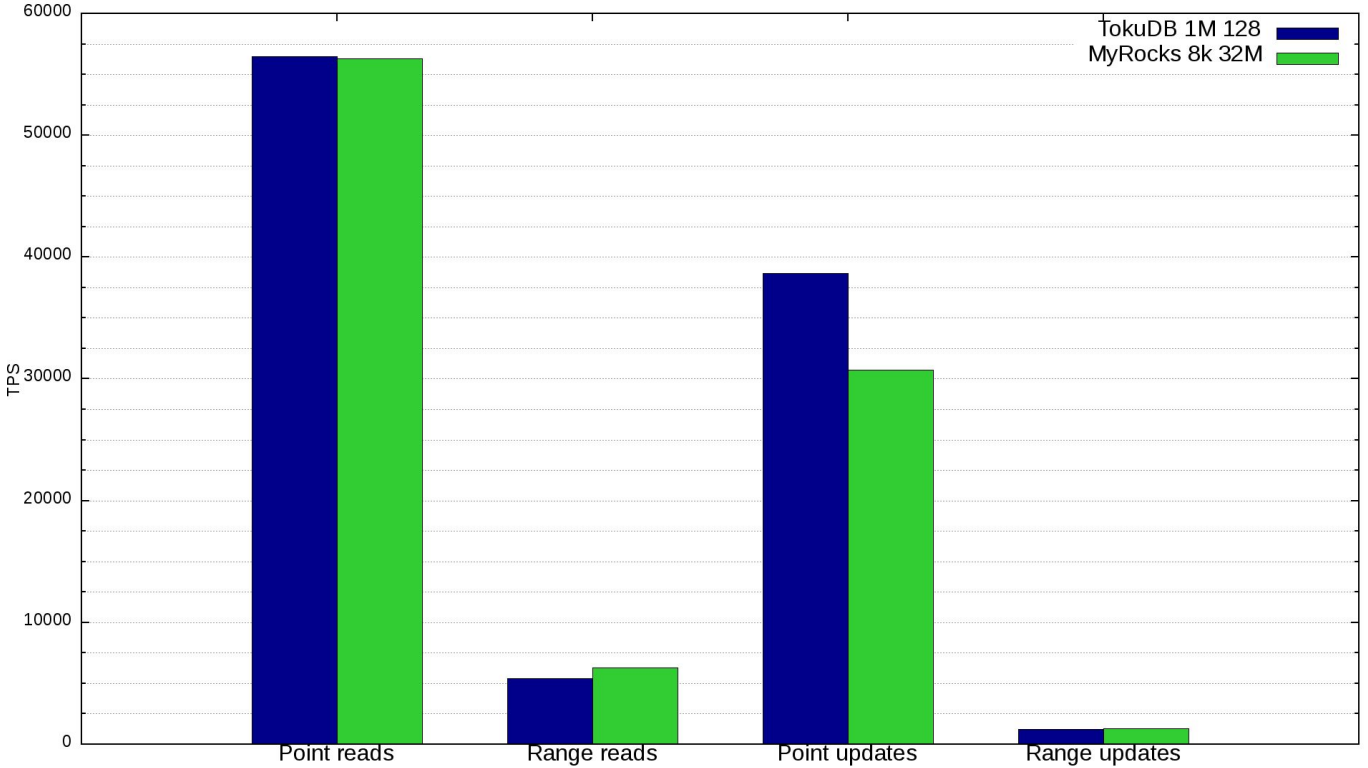


PERCONA

TokuDB vs MyRocks: benchmarking, second case

- NVME, ~100G DB size, 4G - DB cache, 4G - fs cache
- 48 HW threads, 40 sysbench threads
- TokuDB: block size 1M, fanout 128
- MyRocks: block size 8k, first level size 32M
    - 5 levels - the same amount as for 400G table size

TokuDB vs MyRocks: TPS second case

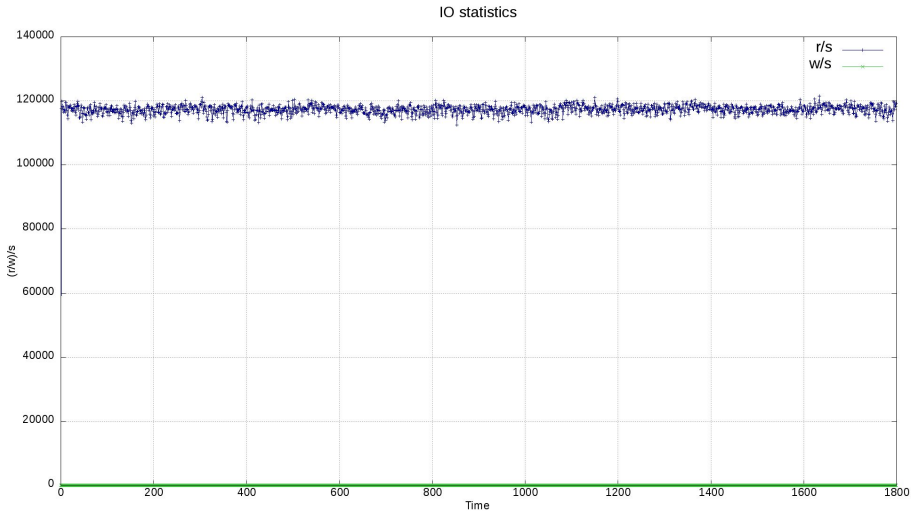


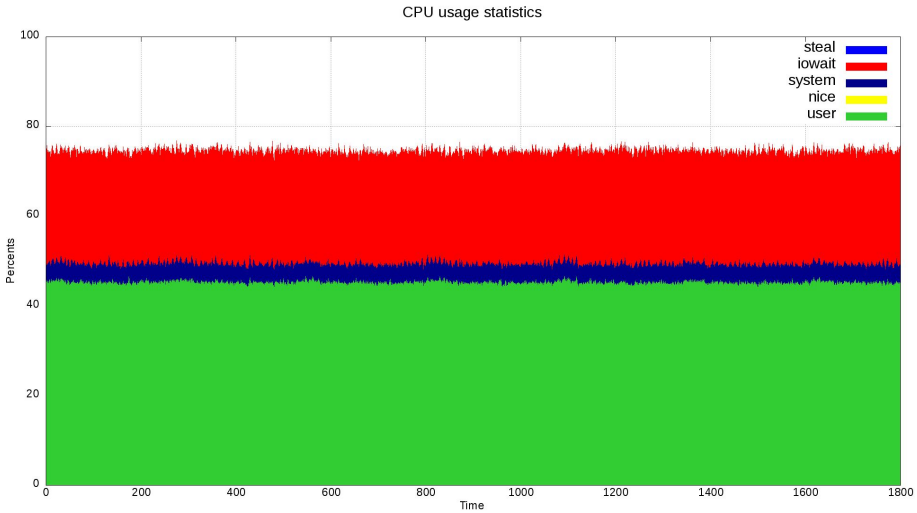
PERCONA

TokuDB vs MyRocks: TPS summary

- Updates cause reads, TokuDB fanout affects read performance
- The less TokuDB block size, the more effective caching, the less IO’s
- TokuDB range reads looks suspicious
- MyRocks - as we preserved the same amount of levels and the same cache/persistence storage balance, the TPS is almost the same as on previous test

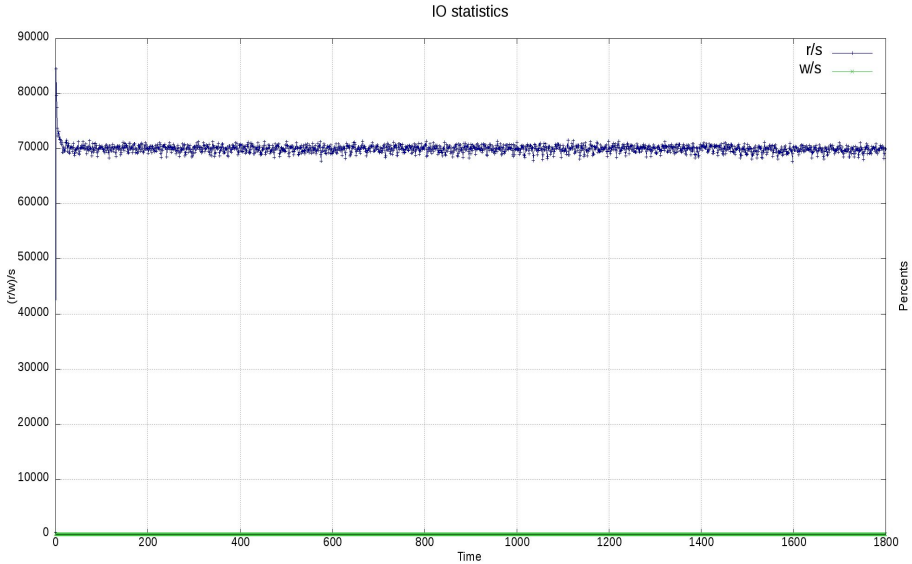
MyRocks: range reads io and cpu

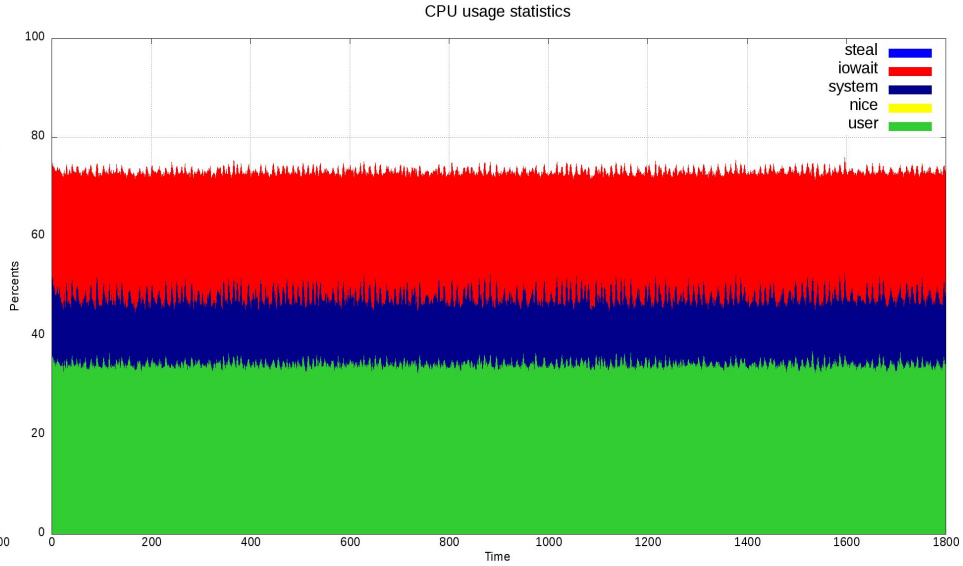




PERCONA

TokuDB: range reads io and cpu





PERCONA

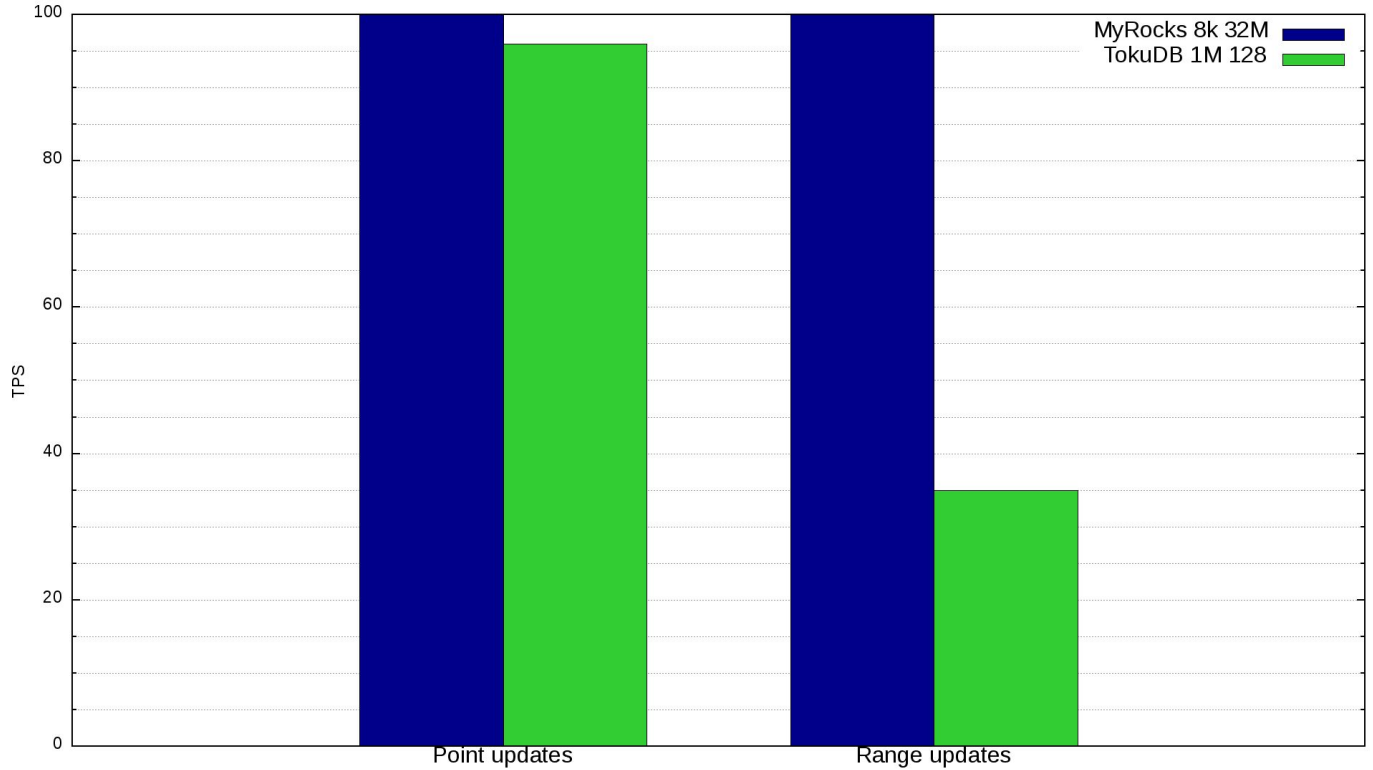
TokuDB: range reads performance

PFS is coming soon...

TokuDB vs MyRocks: write amplification counting

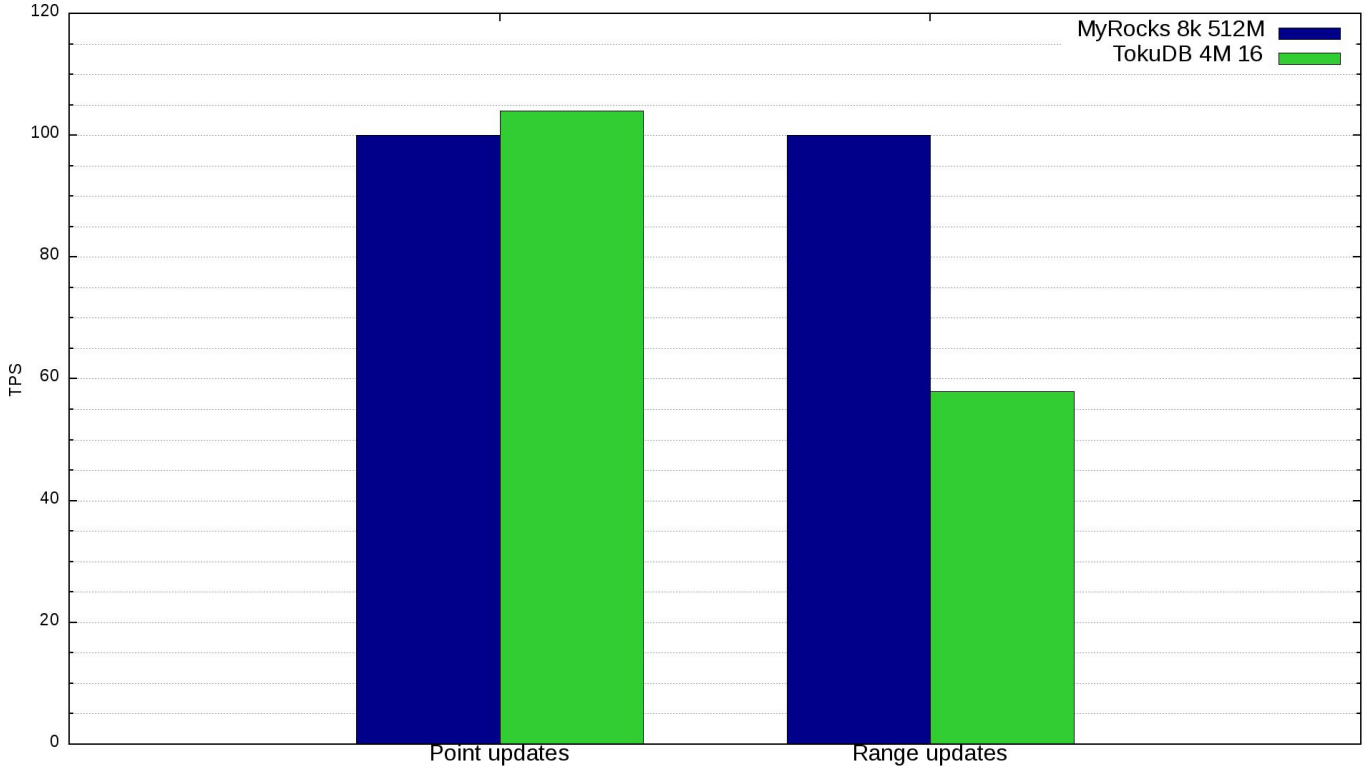
- /proc/diskstats shows blocks written
- Sysbench shows the amount of executed transactions
- WA = K * (blocks written / amount of executed transactions)
- K =((block size) / (row size) + binlog writes)
- Count relative values, let MyRocks WA is 100%, K is diminished

TokuDB vs MyRocks: write amplification



PERCONA

TokuDB vs MyRocks: write amplification

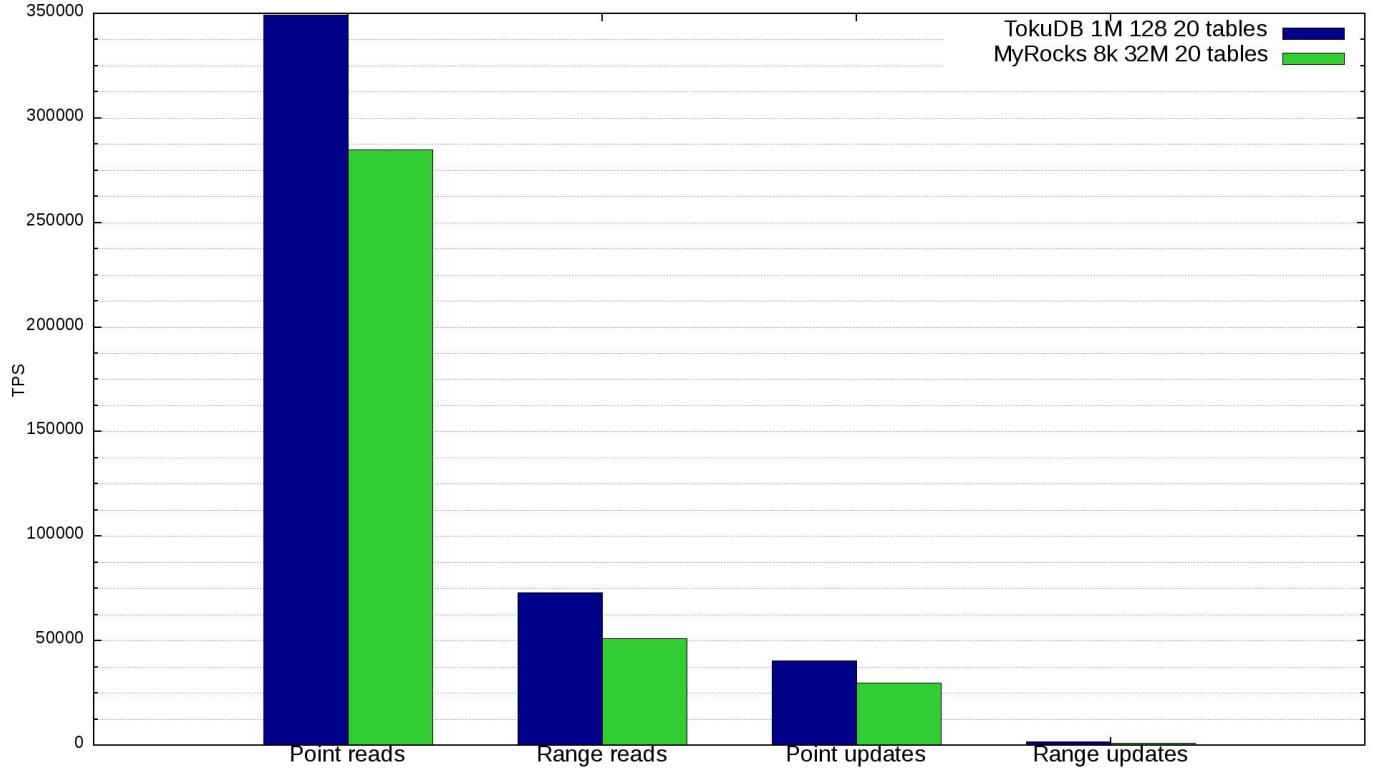


PERCONA

TokuDB vs MyRocks: what about several tables

- NVME, ~100G DB size, 4G - DB cache, 4G - fs cache
- 48 HW threads, 40 sysbench threads
- TokuDB: block size 1M, fanout 128
- MyRocks: block size 8k, first level size 32M
- 20 tables

TokuDB vs MyRocks: TPS, 20 tables

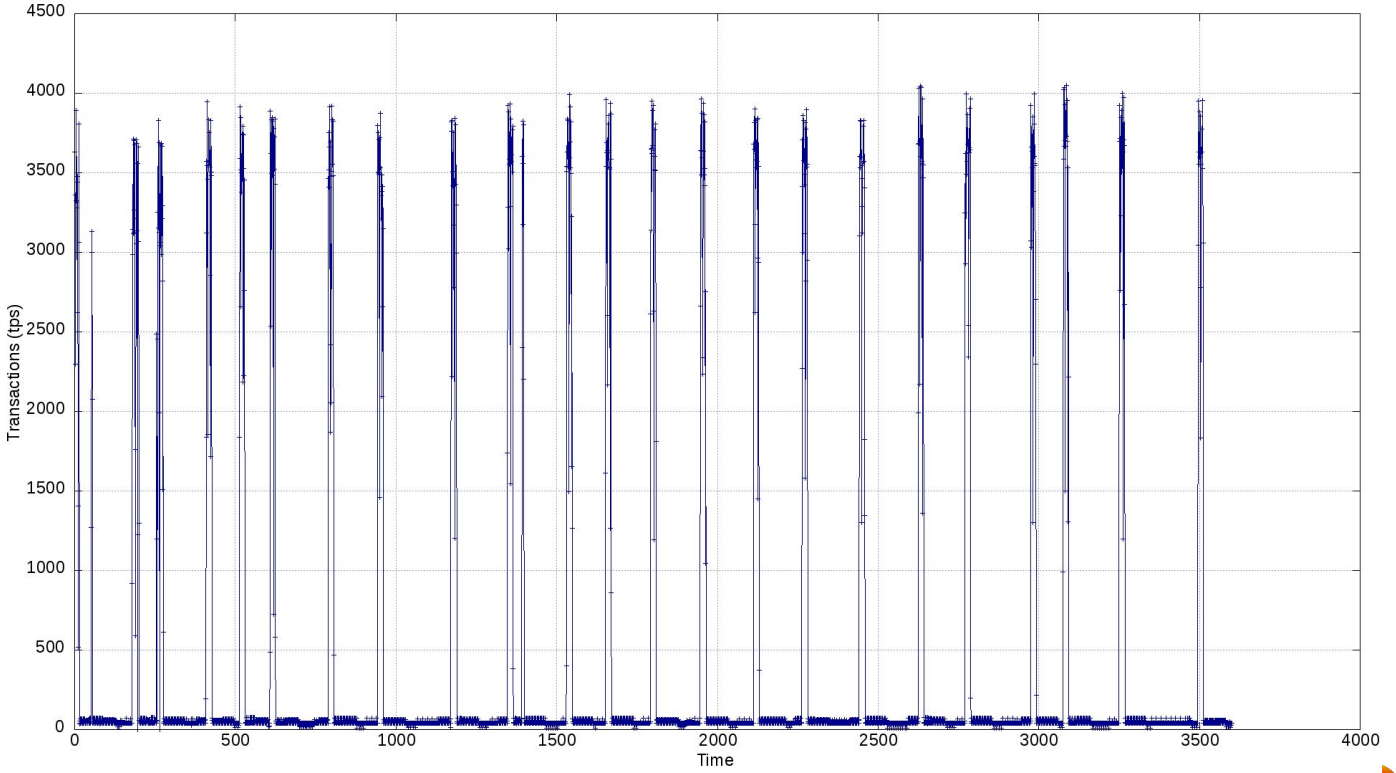


PERCONA

TokuDB vs MyRocks: benchmarking, third case

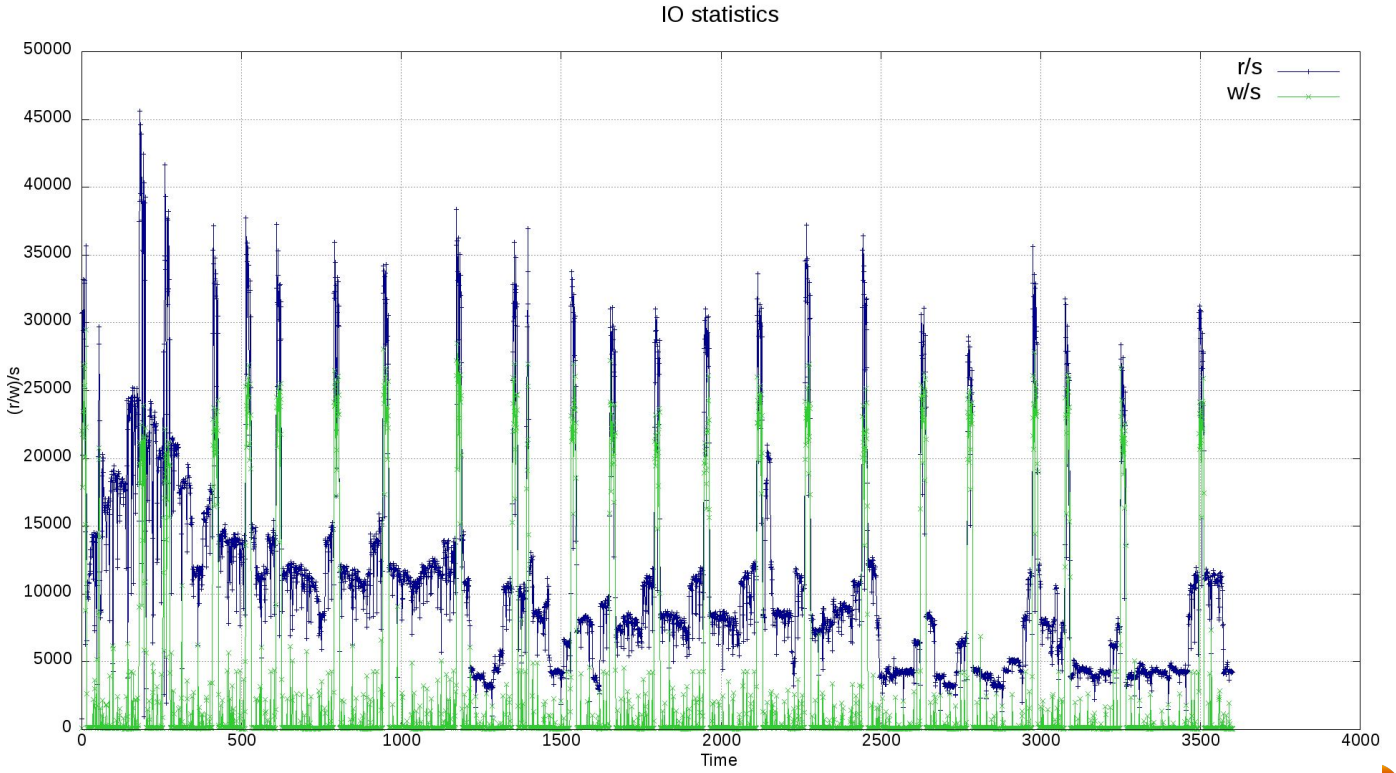
- NVME, ~200G compressed table size, 10G - DB cache, 10G - fs cache
- 48 HW threads, 40 sysbench threads
- TokuDB: block size 1M, fanout 128
- MyRocks: block size 8k, first level size 512M
  - 5 levels

MyRocks: TPS of range updates



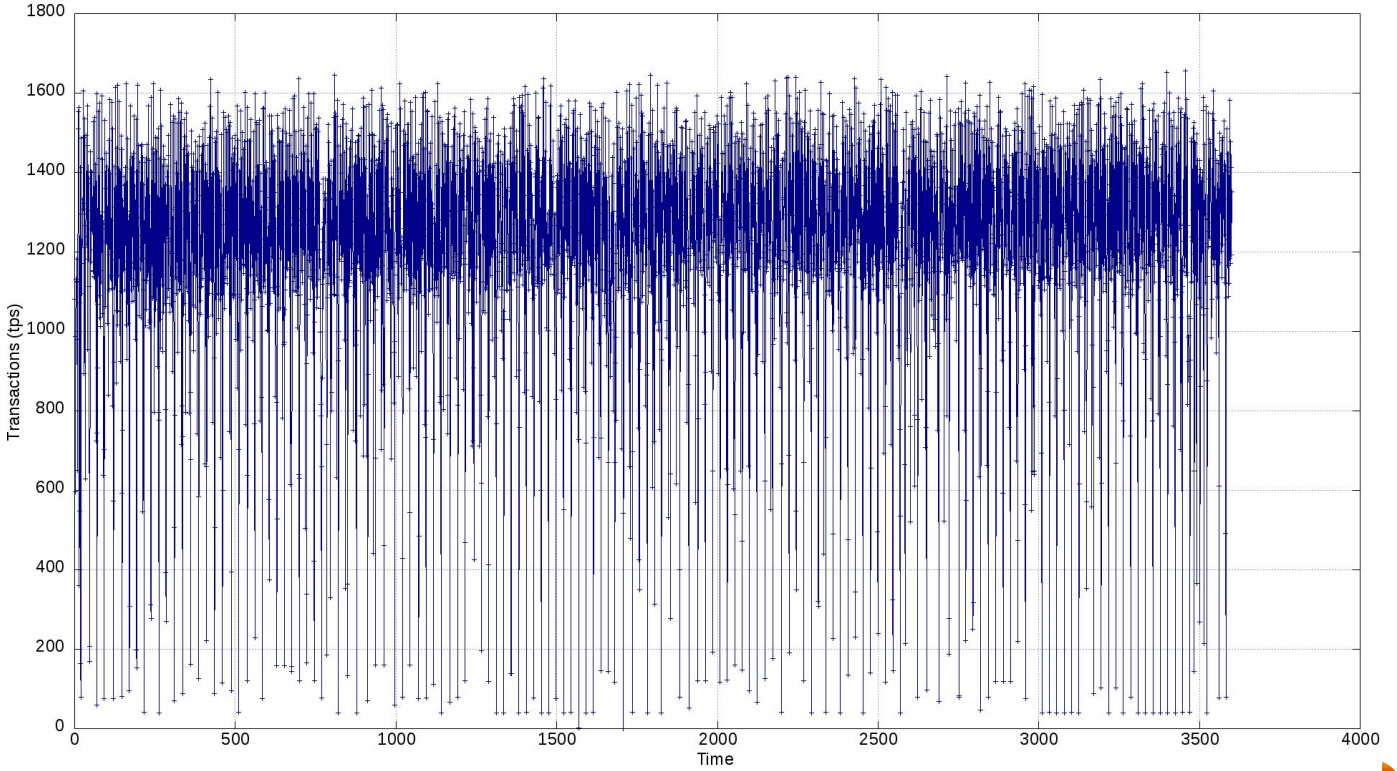
PERCONA

MyRocks: io of range updates



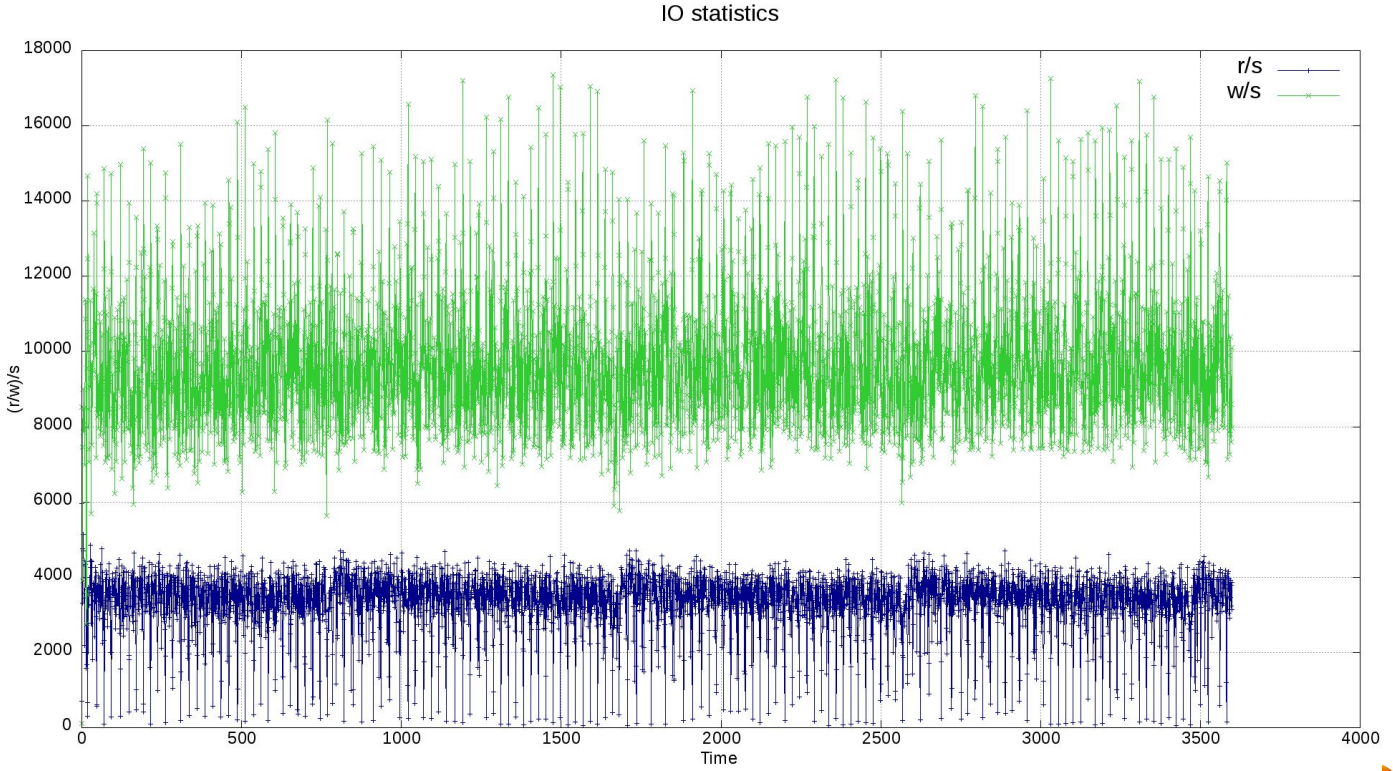
PERCONA

TokuDB: TPS of range updates



PERCONA

TokuDB: io of range updates



PERCONA

TokuDB vs MyRocks: benchmarking, space

For all the benchmarks:

- The initial space TokuDB and MyRocks tables is approximately the same
- TokuDB space doubles after heavy updates
- MyRocks space doubles too but after compaction the allocated space decreases to the initial size + ~10%.

TokuDB vs MyRocks: benchmarking, what to try more?

- Play with smaller TokuDB block sizes
- Play with different amount of levels, block sizes etc. in Rocks
- Test it on rotated disks
- Test for timed series load
- Test for reads after updates/deletes
- Test for inserts
- Test for mixed load
- Test for secondary indexes performance

TokuDB vs MyRocks: features

TokuDB

- Clustered indexes
- Online index
- Gap locks

MyRocks

- Different settings for MemTable
- ColumnFamilies
- Per-level compression settings

DATABASE PERFORMANCE
MATTERS