Pythian

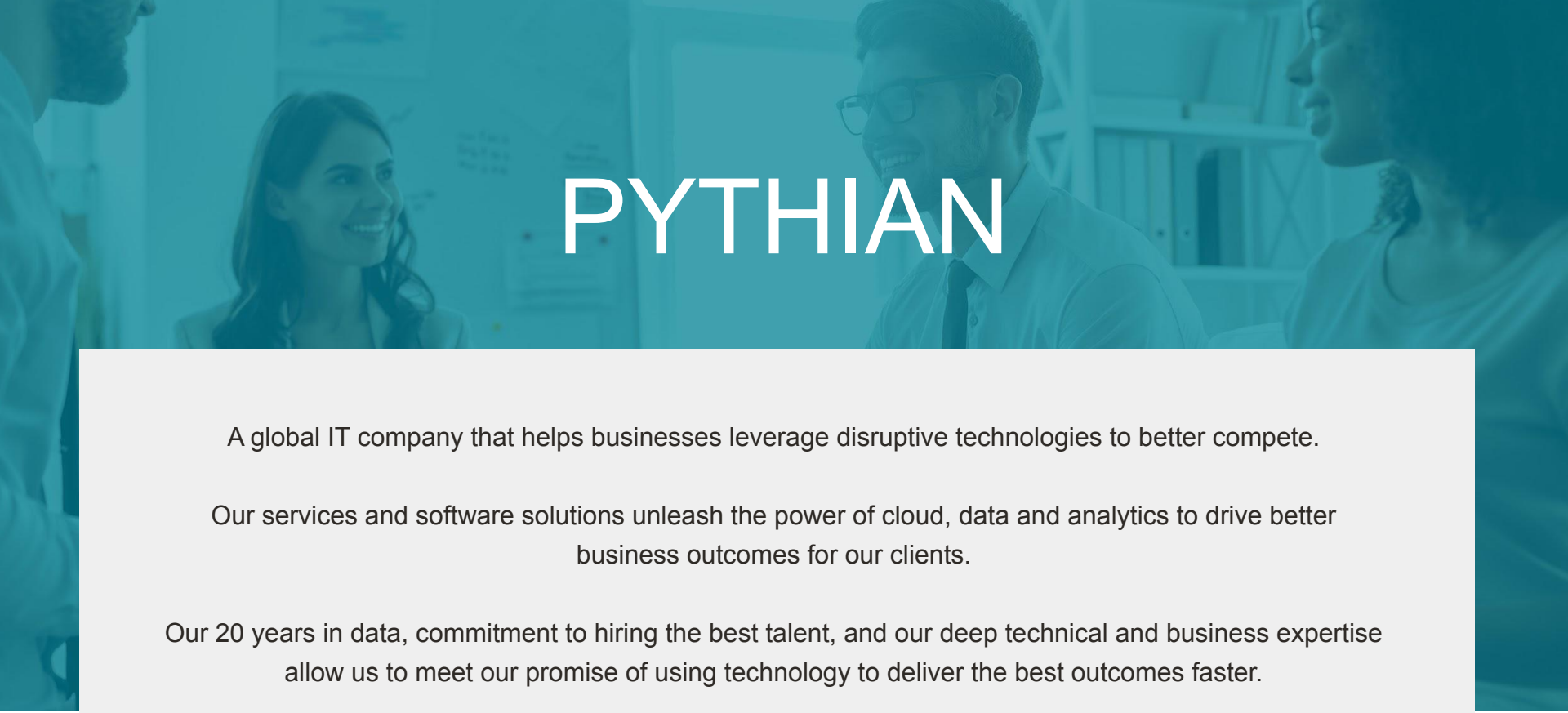
L♥VE YOUR DATA

MyRocks 101

Presented By: Peter Sylvester

About the presenter

- Joined Pythian team in January 2015
- Worked with MySQL since 2008 (5.0) in both DBA and DBD roles
- Originally worked as SQL Server DBA
- Is originally from Detroit, but currently lives in the Greater Toronto Area.
- Social media
  - Twitter = @PeterTheDBA
  - Linkedin = https://www.linkedin.com/in/petertsylvester



PYTHIAN

A global IT company that helps businesses leverage disruptive technologies to better compete.

Our services and software solutions unleash the power of cloud, data and analytics to drive better business outcomes for our clients.

Our 20 years in data, commitment to hiring the best talent, and our deep technical and business expertise allow us to meet our promise of using technology to deliver the best outcomes faster.

20

Years in Business

400+

Pythian Experts
in 35 Countries

350+

Current Clients
Globally

DIVERSIFIED, BLUE CHIP CLIENT BASE

Media/Information Services	Retail	E-commerce	SaaS	Financial Services
FOX SPORTS	SONOS	Tech Data	Grasshopper	PIRAEUS BANK
NATIONAL GEOGRAPHIC	URBAN OUTFITTERS	freshdirect	mapp	COMMERZBANK
AEG	ARC'TERYX		etouches love your event software	البنك السعودي الفرنسي Banque Saudi Fransi
BBM	ALLSAINTS	Camelot GLOBAL	Entrust Datacard	Paymentus
medianet	FAST RETAILING	rockyou	Ford FordDirect.com	TaxAudit Audits happen. Be prepared.
			CROWDTWIST Loyalty. Engaged.	

MyRocks: Bird’s eye view

- Direct storage access to RocksDB
  - Facebook project for MySQL Community 5.6
  - Initially released May 2012
  - Based on LevelDB
- Log Structured Merge Key Value Store
- Available in **Percona Server 5.7** and MariaDB 10.3 (Beta)

MyRocks: Why is this important?

- Percona announces in December 2018 Percona Server would deprecate the TokuDB engine and recommended use of MyRocks
- RocksDB has been at the center of emerging technologies
  - PingCAP / TiDB
  - CockroachDB
- RocksDB being made available as a storage engine for Cassandra

Basis of presentation

- Information was very difficult to find online, conducted my own research
- Converted my notes into a 7-part blog series on internals, focusing on
  - Data writing into memory
  - Flushing to disk
  - Compaction
  - Compression & Bloom Filters
  - Data Reads
  - Replication
  - Use case considerations

Focused on exposing mechanics by looking as system server and status variables, column family options, information_schema tables, and some performance schema instruments.

Basis of presentation

- What we're covering today
- Mechanics at a high level: data in, data out
  - Data writing into memory / Flushing to disk
  - Compaction
  - Compression & Bloom Filters
  - Data Reads
- Crucial variables and metrics for each section listed above
- Use Case Considerations
- Q&A, time permitting

This is only going to cover the basics. Please see the blog series for full details.

MyRocks: Row -> Key Value

- Key
  - Internal Index ID
  - Explicitly defined primary key
- Value
  - Non-primary key columns

Handy Link: MyRocks record format

MyRocks: Column Families

Multiple tables and secondary indexes can belong to the same column family.

```
CREATE TABLE `t1` (
  `c1` int(10) unsigned NOT NULL AUTO_INCREMENT,
  `c2` char(255) NOT NULL,
  PRIMARY KEY (`c1`) COMMENT 'cf_t1'
) ENGINE=ROCKSDB DEFAULT CHARSET=latin1
```

Recommendation from Facebook team is not to create more than 20 column families.

MyRocks: CF Configuration Considerations

MySQL Configuration Scope

- Global
- Session

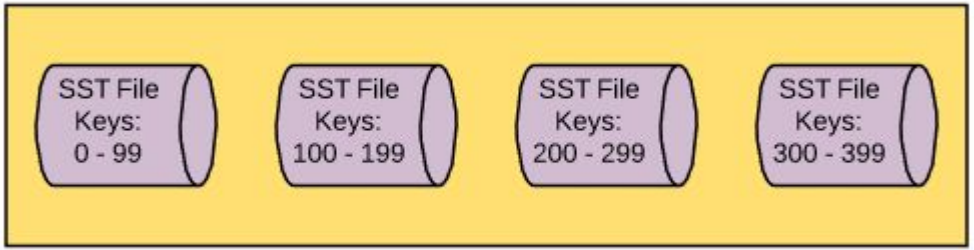
MyRocks Configuration Scope

- Global
- Session
- Column Family Option
  - Typically configured with variable rocksdb_default_cf_options

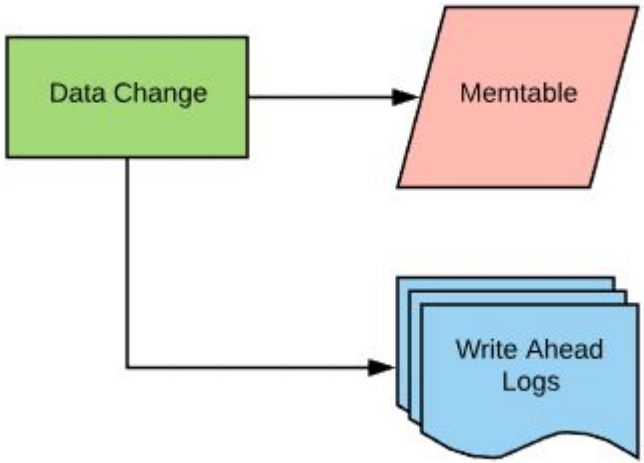
Important distinction when you can have multiple column families each with their own memory caches.

MyRocks: Data Format objective

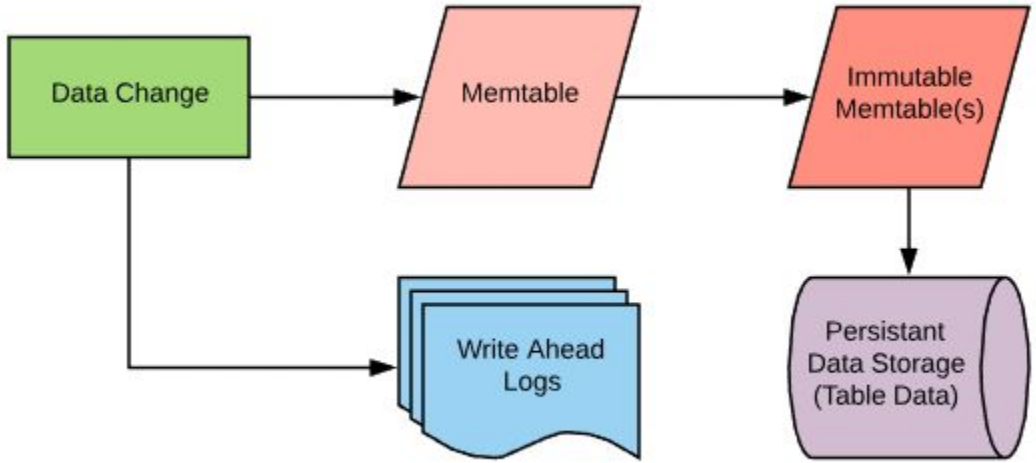
Create a series of data files that each store a non-overlapping range of keys, making for a faster lookup.



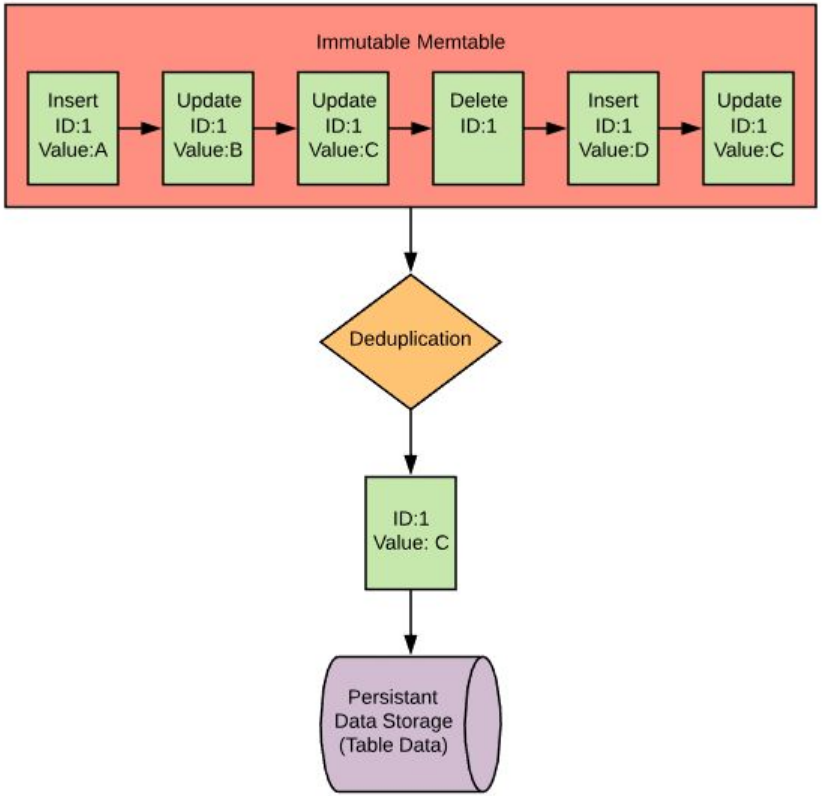
Initial data writing: Start Writing to Memtable



Initial data writing: Memtable is full!



Initial data writing



Initial data writing: Important variables / options

- CF_OPTION write_buffer_size

This is the size of each memtable.

Active tables and immutable memtables are the same size.

Default: 64 Mb

Initial data writing: Important variables / options

- CF_OPTION min_write_buffer_number_to_merge

How many immutable memtables should be created before that first flush to disk will occur.

Impacts flushing rate, flushing size, and also how much memory will be consumed.

Default: 1

Currently this option doesn't appear to change anything. I have opened up a bug with Percona in regard to this. Bug PS-5437

Initial data writing: Important variables / options

- Rocksdb_db_write_buffer_size

Maximum amount of memory that can be consumed by active memtables across all column families

Default: 0 (no limit)

Highly recommended to set this to a non-zero value.

Initial data writing: Important variables / options

- Rocksdb_max_total_wal_size

Maximum amount of disk space that can be consumed by active write ahead logs across all column families

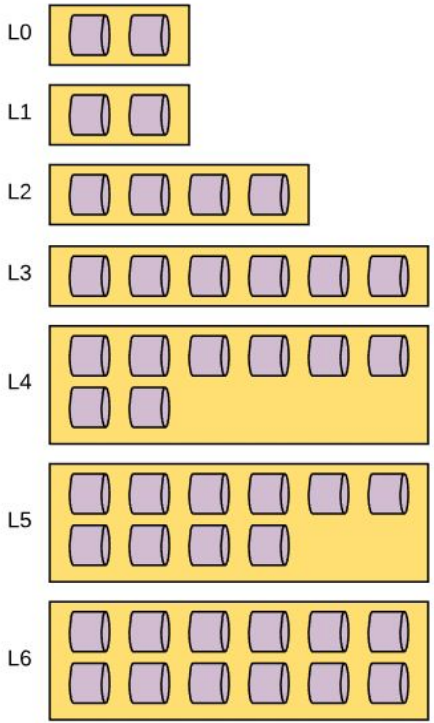
Default: 0 (no limit)

Highly recommended to leave this at 0. Hitting the limit will force a flush of all memtables so that a new WAL can be created.

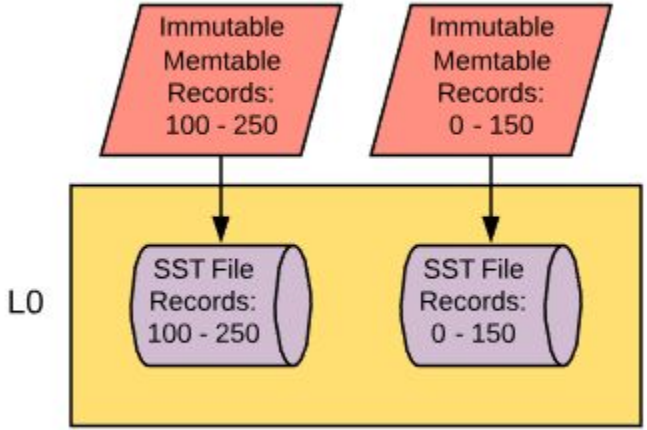
Initial data writing: Associated Metrics

- System status variables (show global status output)
  - Rocksdb_memtable_total
    - Current amount of memory currently being consumed by memtables
  - Rocksdb_stall_memtable_limit_slowdowns / stops
    - Number of times MyRocks has has to stop or throttle writes due to hitting the maximum number of allowable memtables
- information_schema.ROCKSDB_CFSTATS table
  - CUR_SIZE_ACTIVE_MEM_TABLE
    - The current size of all active memtables per column family
  - NUM_ENTRIES_ACTIVE_MEM_TABLE
    - The number of record changes in the active memtables per column family
  - NUM_ENTRIES_IMM_MEM_TABLES
    - The number of record changes in the immutable memtable(s) per column family

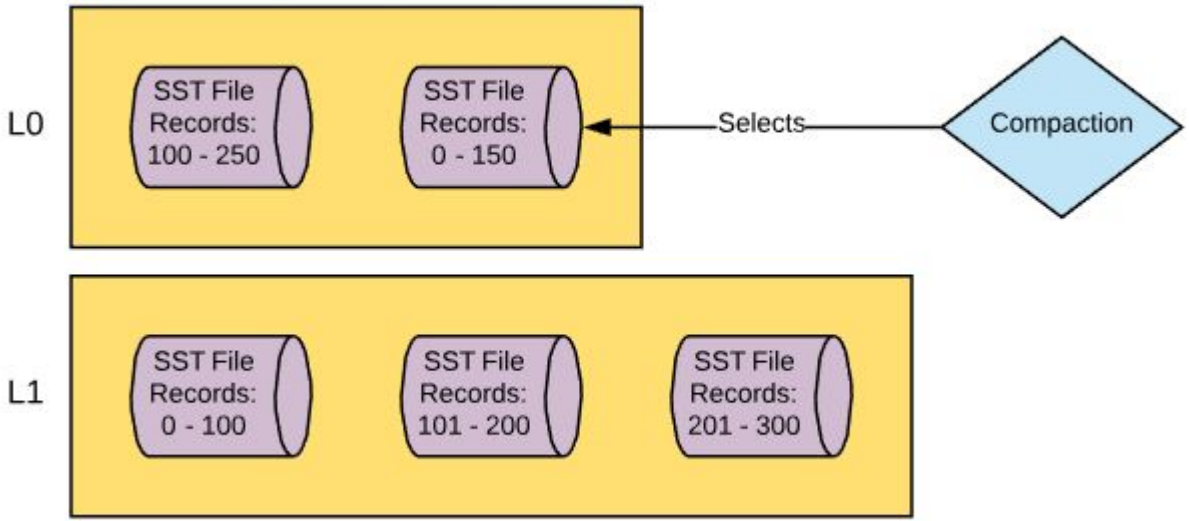
What happens after the initial flush? Compaction!



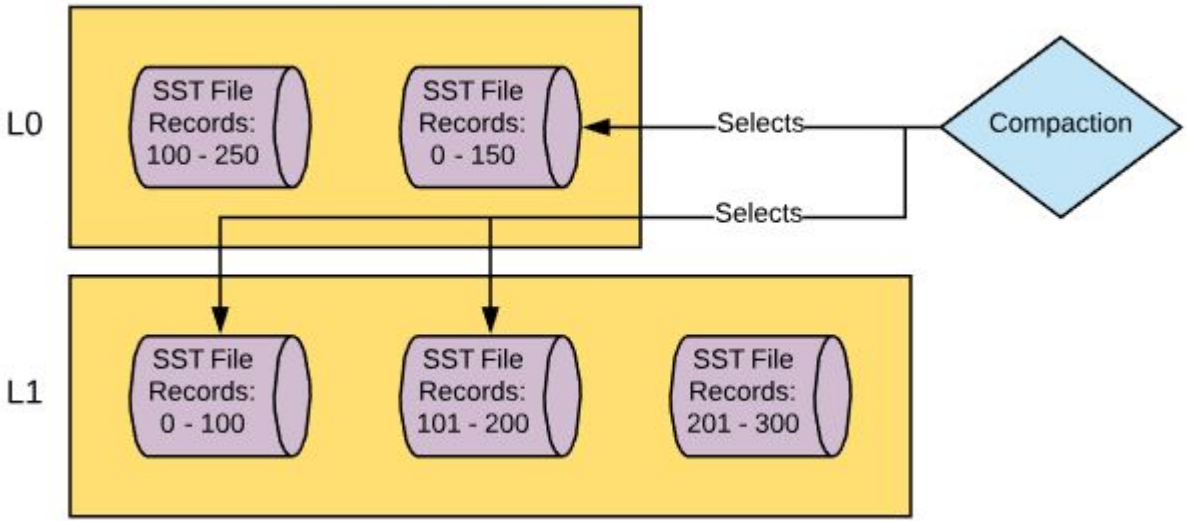
Compaction



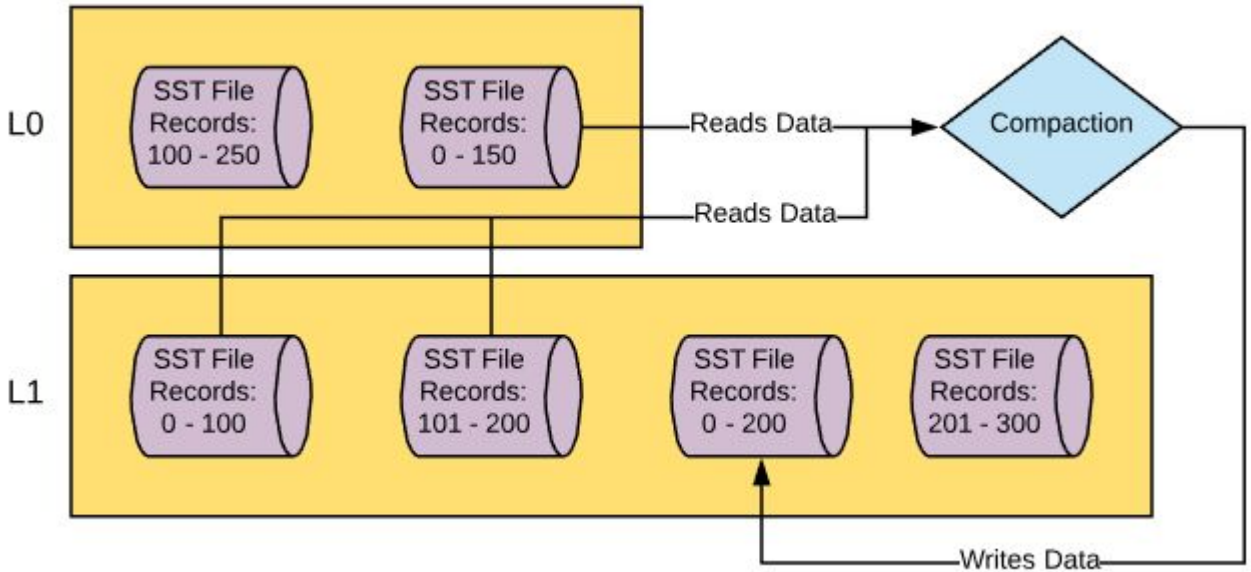
Compaction



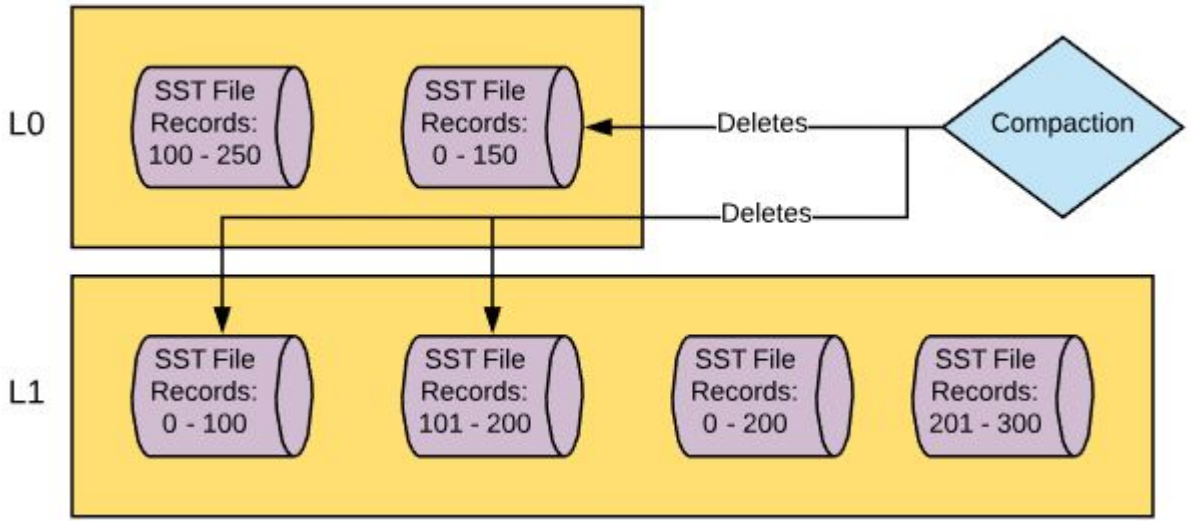
Compaction



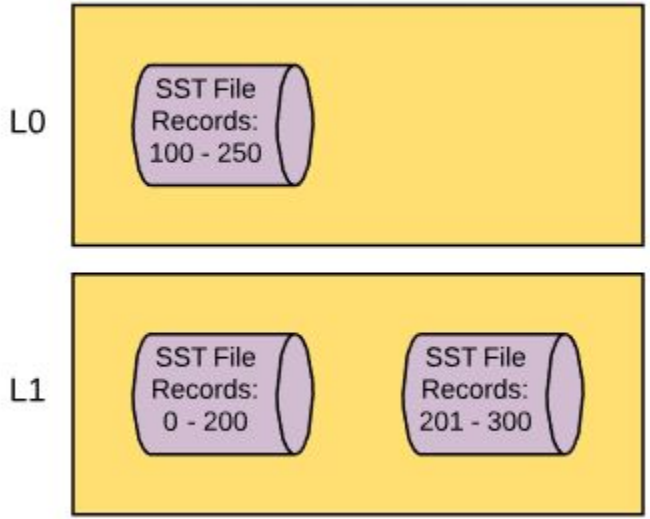
Compaction



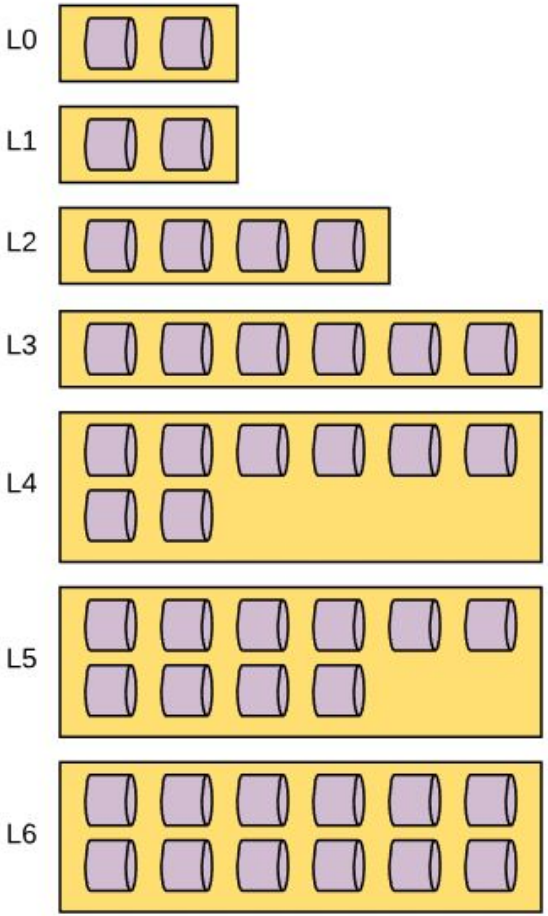
Compaction



Compaction



Compaction



Compaction: Important variables / options

- CF_OPTION Level0_file_num_compaction_trigger

The maximum number of files in L0 before compaction to L1 is triggered

Default: 4 (256Mb max)

Compaction: Important variables / options

- CF_OPTION Max_bytes_for_level_base

The maximum number of bytes in L1 before compaction to L2 is triggered

Default: 268435456 (256Mb)

Compaction: Important variables / options

- CF_OPTION Max_bytes_for_level_multiplier

Make N compaction layer X times larger than N-1

Default: 10

L2: 2.5 Gig

L3: 25 Gig

L4: 250 Gig

L5: 2.44 Tb

L6: 24.41 Tb

Compaction: Important variables / options

- CF OPTION Num_levels

Maximum number of compaction layers

Default: 7 (L0 - L6)

Compaction: Important variables / options

- CF_OPTIONS Target_file_size_base & Target_file_size_multiplier

This sets the size of the files at each layer of compaction

L0 = Size of deduplicated memtables

L1 = Size of Target_file_size_base

L2+ = Size of N-1 layer multiplied by Target_file_size_multiplier

Default: 64Mb / 1 (respectively). All files will be 64Mb

Compaction: Important variables / options

- Rocksdb_max_background_jobs

Number of threads can be used for table flushing and compaction

Used to be seperate variables

Default: 2

I would recommend increasing given the intended multithreaded approach to compaction in RocksDB

Compaction: Important variables / options

- Rocksdb_max_subcompactions

Number of ‘subthreads’ used to support each compaction thread

Default: 1

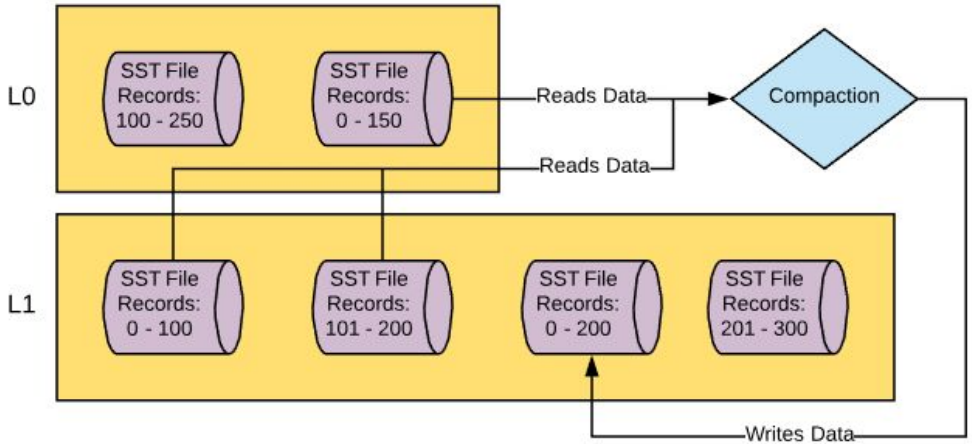
Given the single threaded nature of L0 -> L1 compaction, I would recommend increasing this variable.

Compaction: Associated Metrics

- System status variables (show global status output)
  - Rocksdb_stall_l0_file_count_limit_slowdowns / stop
    - Number of times MyRocks has has to stop or throttle writes due to L0 being close to full since last MySQL restart
    - Check out this wiki entry on write stalling for a lot more info
- information_schema.ROCKSDB_CFSTATS table
  - COMPACTION_PENDING
    - Shows the current number of pending compaction requests
- Additional information_schema tables
  - ROCKSDB_COMPACTION_STATS
  - ROCKSDB_DDL
  - ROCKSDB_INDEX_FILE_MAP

We've got several copies of our data… how do we manage this effectively?

- Compression
- Bloom filters



Bloom Filters

- Space efficient data structure used to assist with the determination of set membership

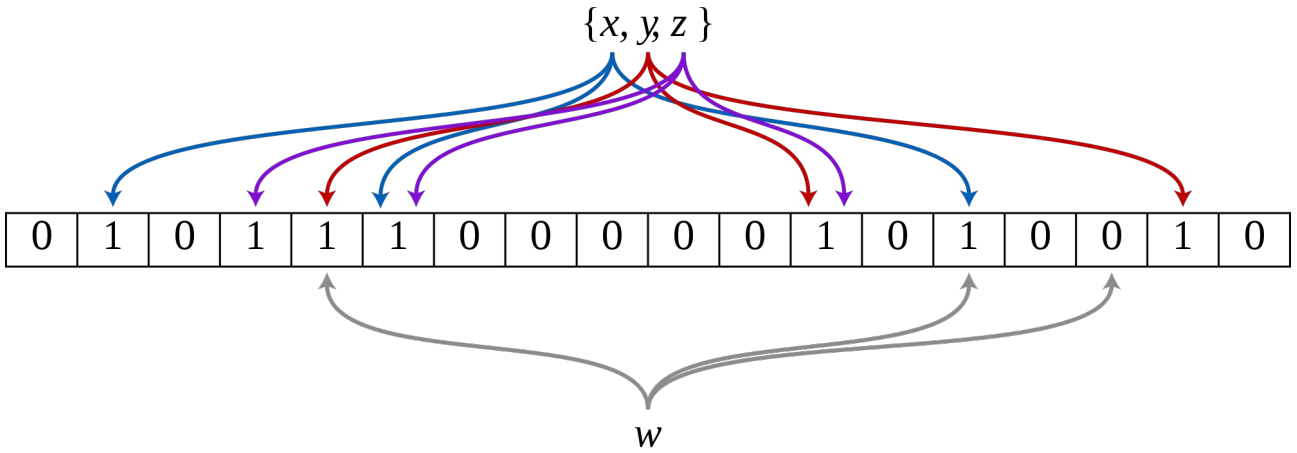


Image Credit: Wikipedia

Bloom Filters: Important variables / options

- CF_OPTION Block_based_table_factory: Filter_policy

Enables bloom filtering

Default: NULL (disabled)

Has a strange configuration requirement

```
rocksdb_default_cf_options=block_based_table_factory={filter_policy=\
bloomfilter:10:false}
```

Bloom Filter: Associated Metrics

- System status variables (show global status output)
  - Rocksdb_bloom_filter_useful
    - Number of times a bloom filter resulted in the avoidance of a data read

Compression

- Can use multiple forms of compression
  - kZSTD
  - kXpressCompression
  - kLZ4HCCompression
  - kLZ4Compression
  - kBZip2Compression
  - kZlibCompression
  - kSnappyCompression

Compression

```
[root@centos7-1 .rocksdb]# cat ./LOG | grep -A 10 "Compression algorithms supported"
2019/03/01-09:28:38.437724 7ff6cfd44880 Compression algorithms supported:
2019/03/01-09:28:38.437727 7ff6cfd44880         kZSTDNotFinalCompression supported: 1
2019/03/01-09:28:38.439318 7ff6cfd44880         kZSTD supported: 1
2019/03/01-09:28:38.439324 7ff6cfd44880         kXpressCompression supported: 0
2019/03/01-09:28:38.439326 7ff6cfd44880         kLZ4HCCompression supported: 1
2019/03/01-09:28:38.439327 7ff6cfd44880         kLZ4Compression supported: 1
2019/03/01-09:28:38.439329 7ff6cfd44880         kBZip2Compression supported: 0
2019/03/01-09:28:38.439330 7ff6cfd44880         kZlibCompression supported: 1
2019/03/01-09:28:38.439332 7ff6cfd44880         kSnappyCompression supported: 0
2019/03/01-09:28:38.439339 7ff6cfd44880 Fast CRC32 supported: Supported on x86
```

Compression

- Compresses
  - 3 - 4x better than InnoDB uncompressed
  - 2x better than InnoDB compressed
- Can be applied...
  - Per column family
  - Per compaction level
- Recommended configuration
  - No compression at L0 & L1 so there is as little overhead with initial flushes and first compaction. Remember, L0 -> L1 compaction is single threaded
  - Moderate compression for >= L2, kLZ4
  - High compression for bottom most compaction layer L6, kZSTD

Compression: Important variables / options

- CF_OPTION Compression
- CF_OPTION Bottommost_compression
- CF_OPTION Compression_per_level

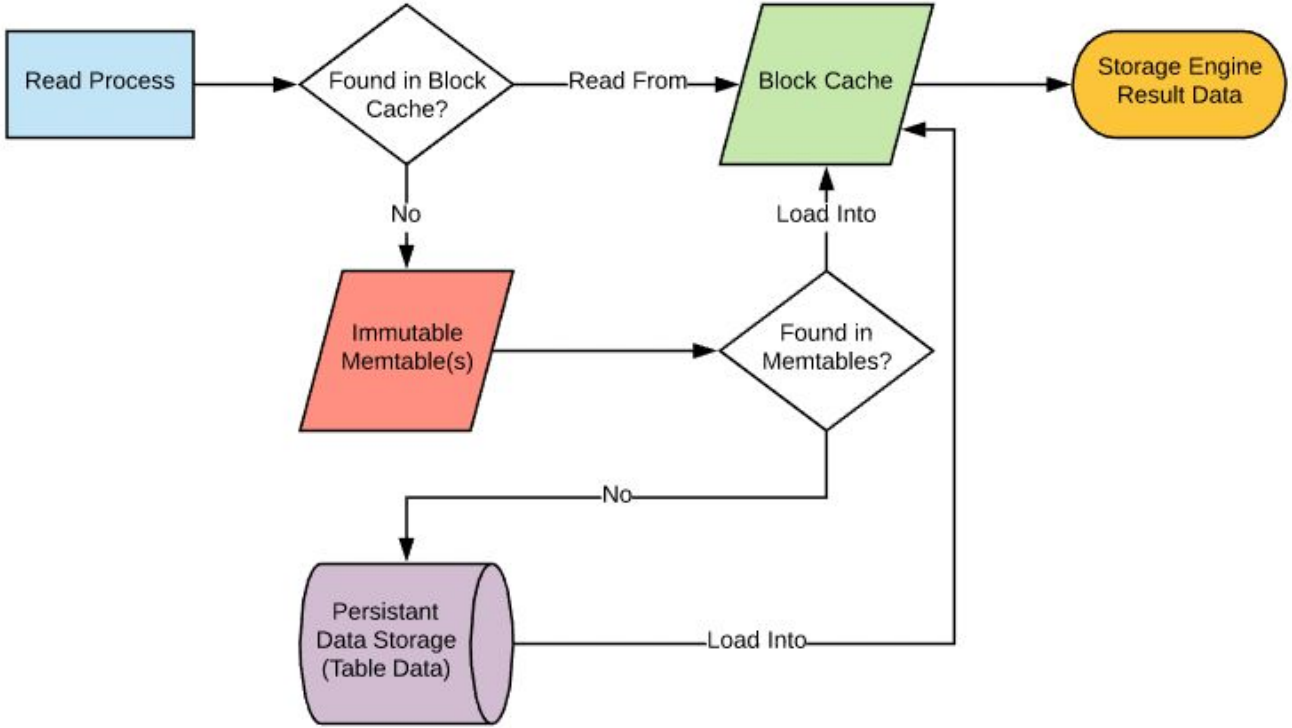
Allow for compressing at various levels of compaction

Default: NULL (disabled)

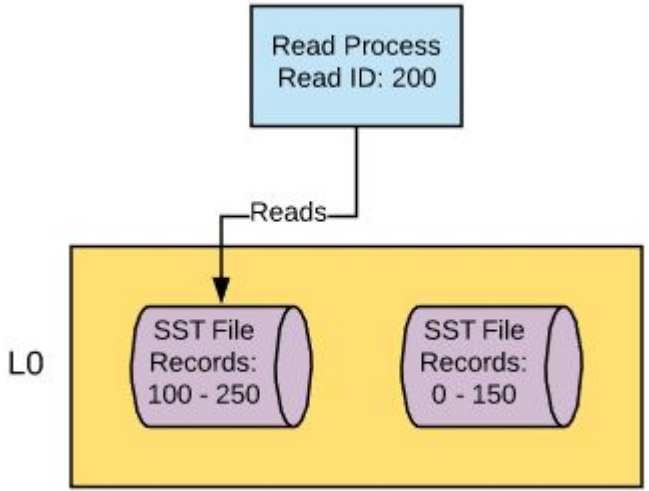
```
--options
compression=kLZ4Compression;
bottommost_compression=kZSTD;
compression_per_level=kNoCompression:kNoCompression:kLZ4Compression:kLZ4
Compression:kLZ4Compression:kLZ4Compression:kZSTD;
```

Compression: A word from your speaker

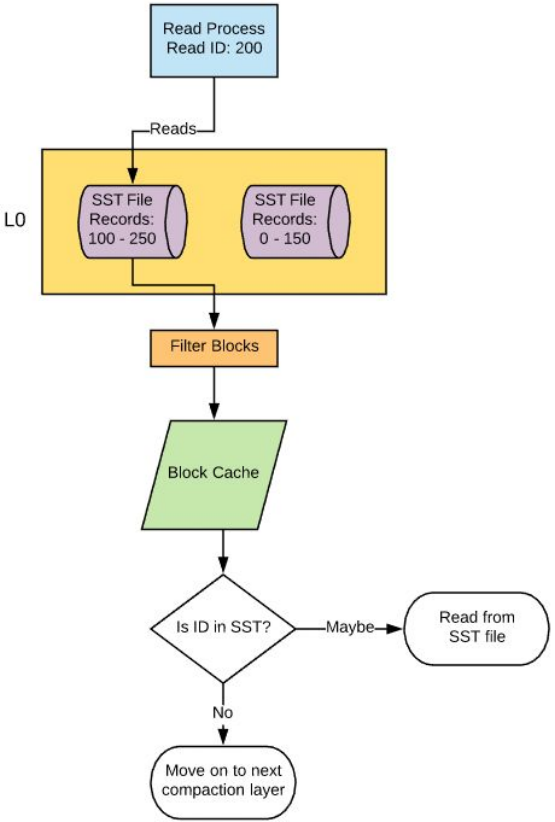
Data Reading



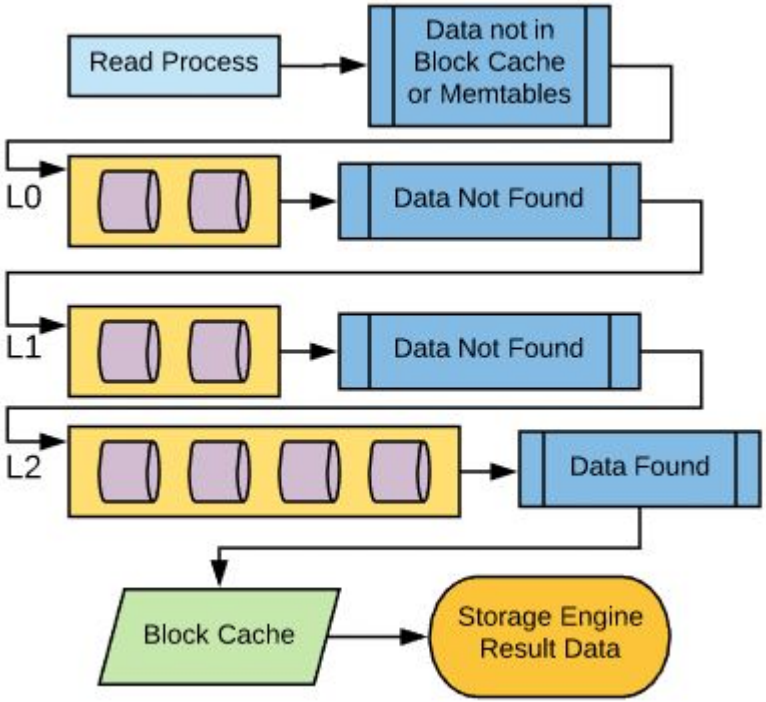
Data Reading



Data Reading



Data Reading



Data Reads: Important variables / options

- Rocksdb_block_cache_size

Primary data cache for reads

Default: 512Mb

Recommended to be 60 - 75% of available system physical memory

Can be disabled temporarily using Rocksdb_skip_fill_cache dynamically

Data Reads: Important variables / options

- Rocksdb_sim_cache_size

Provides hit / miss ratio if the block cache was the size of Rocksdb_sim_cache_size. Only costs 2% of the designated value.

Default: 0 (disabled)

Ever wondered how much larger your read cache would have to be if you wanted a better hit rate? Ever get a constant 99.9% hit rate and wonder how much memory could be used for another part of the database engine?

Disadvantage: Non-dynamic variable

Data Read: Associated Metrics

- System status variables (show global status output)
  - Rocksdb_block_cache_data_hit / Rocksdb_block_cache_data_miss
    - Block cache hit ratio
  - Rocksdb_get_hit_l0 / Rocksdb_get_hit_l1 / Rocksdb_get_hit_l2_and_up
    - Hit rate of various compaction layers. Are you hitting frequently updated data?
  - Rocksdb_bytes_read / Rocksdb_block_cache_bytes_read
    - How much data did I have to pull from disk?
- SHOW ENGINE ROCKSDB STATUS
  - Rocksdb.sim.block.cache.hit
  - Rocksdb.sim.block.cache.miss

MyRocks Use Case Considerations

Ok, I know how it works, but is it right for me?
^

MyRocks Use Case Considerations: Advantages

- Compression
  - Can be configured all the way down to the compaction layer for each column family
  - Can use more or less aggressive compression, you're not stuck with a single compression algorithm
  - Remember that there is more to to a storage engine than just how well it compresses!

MyRocks Use Case Considerations: Advantages

- Write Optimized
  - Compaction = Deferred write amplification

MyRocks: Get the data in the system initially as fast as possible and then organize it later

vs

InnoDB: Organize the data on entry to evenly optimize for read requests

MyRocks Use Case Considerations: Advantages

- Better performance when working with active data sets that are larger than the amount of available system memory

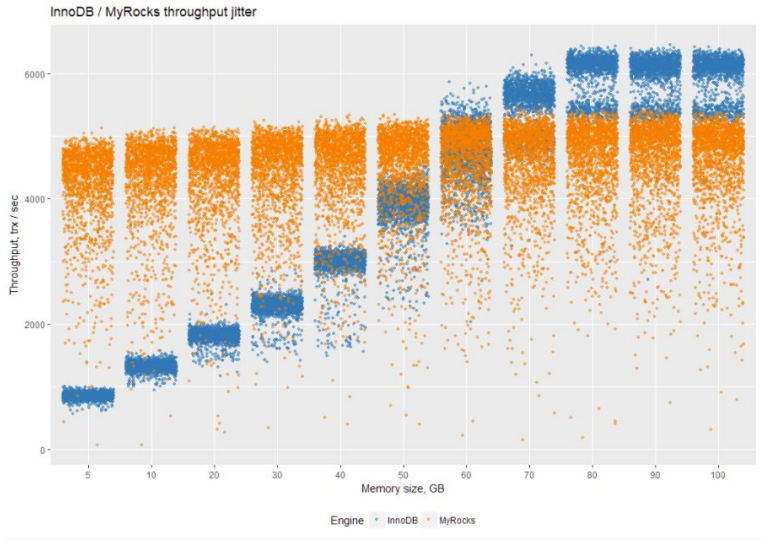


Image Credit: Percona / Vadim Tkachenko

MyRocks Use Case Considerations: Advantages

- Backups

Percona xtrabackup supports MyRocks AND InnoDB in its 8.0.6 release

PERCONA
XtraBackup

MyRocks Use Case Considerations: Disadvantages

- Range Lookups
  - Bloom filters are oriented for point lookup
  - Prefix bloom filters are available
  - RocksDB does have better scanning capabilities. Read more about it in this blog post on why CockroachDB selected RocksDB to be their underlying storage engine.

MyRocks Use Case Considerations: Disadvantages

- Reduced MySQL functionality
  - No Online DDL
  - No Foreign Keys
  - No Transportable Tablespaces
  - No Select for update when using repeatable read isolation level

MyRocks Use Case Considerations: When to use it?

- A large OLTP dataset where your active data set doesn’t fit into memory
- Write intensive workloads
- High concurrency reads that don’t filter on range

Conclusion

- What we're covering today
- Mechanics at a high level: data in, data out
  - Data writing into memory / Flushing to disk
  - Compaction
  - Compression & Bloom Filters
  - Data Reads
- Crucial variables and metrics for each section listed above
- Use Case Considerations

Questions?

Thank You

- Peter Sylvester
Internal Principal Consultant @ Pythian
sylvester@pythian.com

@PeterTheDBA on Twitter
LinkedIn