ORACLE

Increasing your performance

MySQL Enterprise Thread Pooling

Dale Dasker
Manager, MySQL Solution Engineering
dale.dasker@oracle.com
August 24, 2023

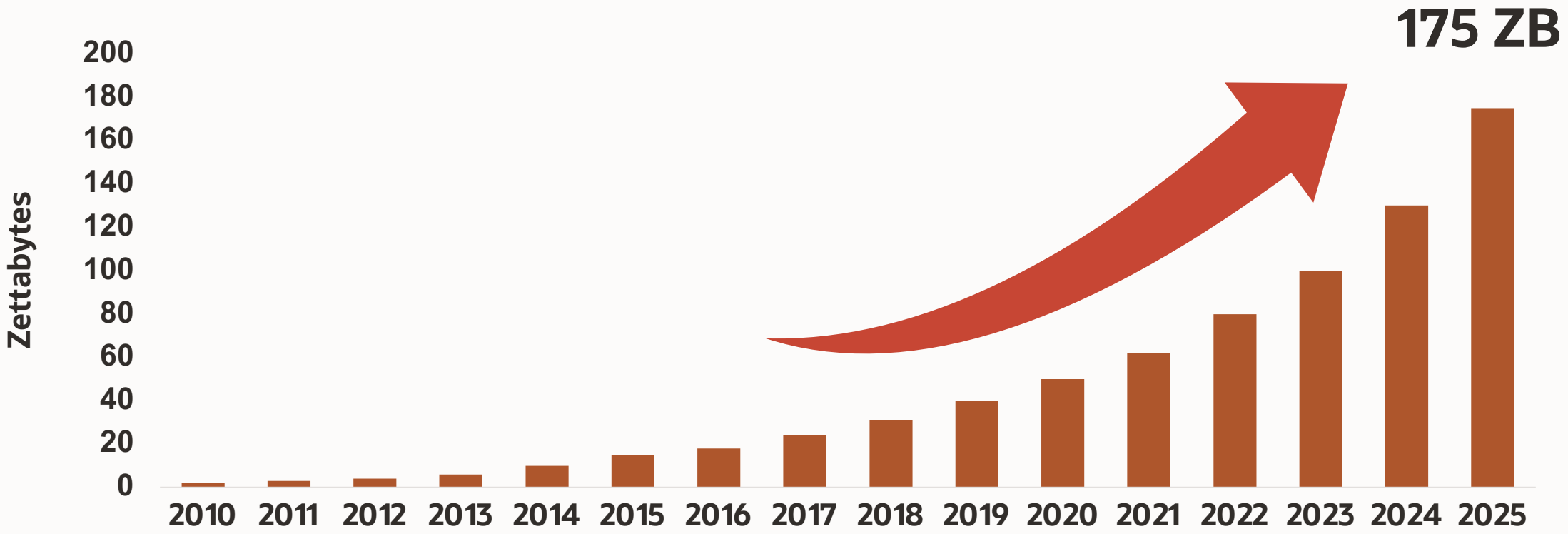
Agenda

Achieve Increased Throughput and Scalability with Enterprise Thread Pooling

- Thread Pooling Overview
- Sysbench Overview
- Setup and Installation of:
  - Enterprise Thread Pool
  - sysBench

Why?

Global Datasphere



MySQL Version / CPU Support Evolution

2008

MySQL 5.0
Up to 4 Cores

2009

MySQL 5.1
(InnoDB Plugin)
Up to 16 Cores
(Sun Micro)

2010

MySQL 5.5
Up to 32 Cores
(Oracle)

2012

MySQL 5.6
Up to 48 Cores

2015

MySQL 5.7
72 Cores

2018

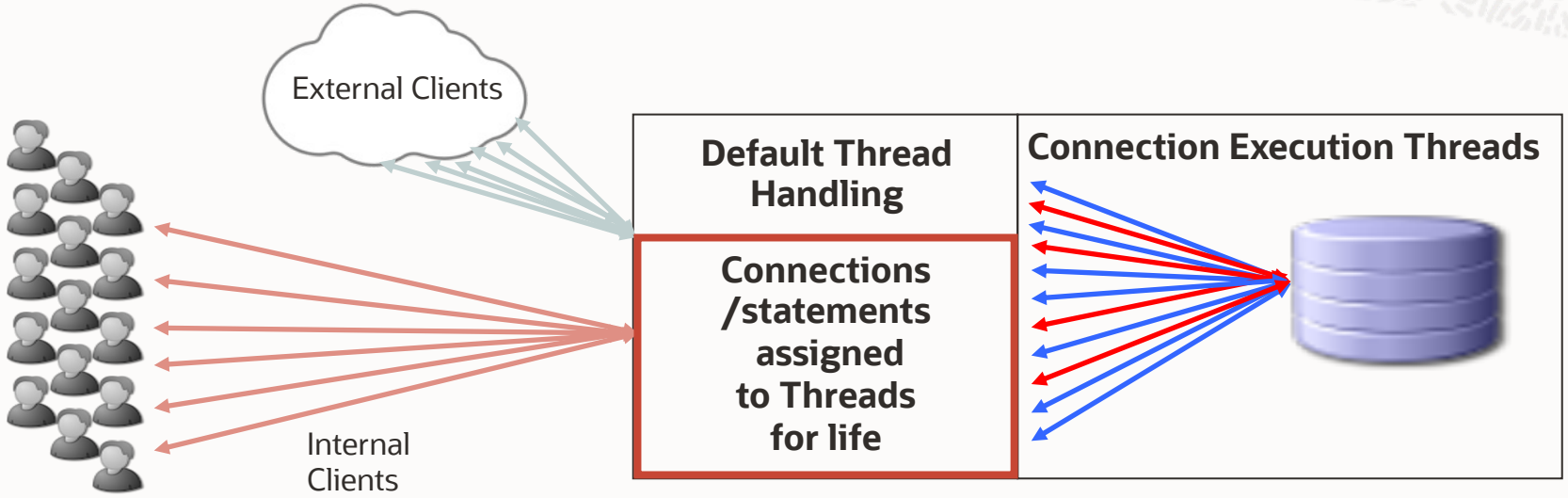
MySQL 8.0
96 Cores

Thread Pool: Problem Definition

- Current mapping from connection to thread is one thread per connection.
  - Memory usage increases
  - More Cache Misses
  - Higher contention on “hot spots”
- Write-Intensive workloads more affected

MySQL Enterprise Thread Pool

Default Connection Behavior



- Connections assigned to 1 thread for the life of the connection, same thread used for all statements
- No prioritization of threads, statement executions
- Many concurrent connections = many concurrent execution threads to consume server memory, limit scalability

Thread Pool Solution

- Wait for execution of a query until the MySQL Server has sufficient CPU and memory resources to execute it.
- Prioritize queries on connections that have an ongoing transaction.
- Split threads into groups individually handled to avoid making the solution a problem in itself, aim is to manage one active thread per group.
- Avoid deadlocks when queries are stalled or execute for a long time.

Thread Pool Solution: Avoiding Scalability Issue

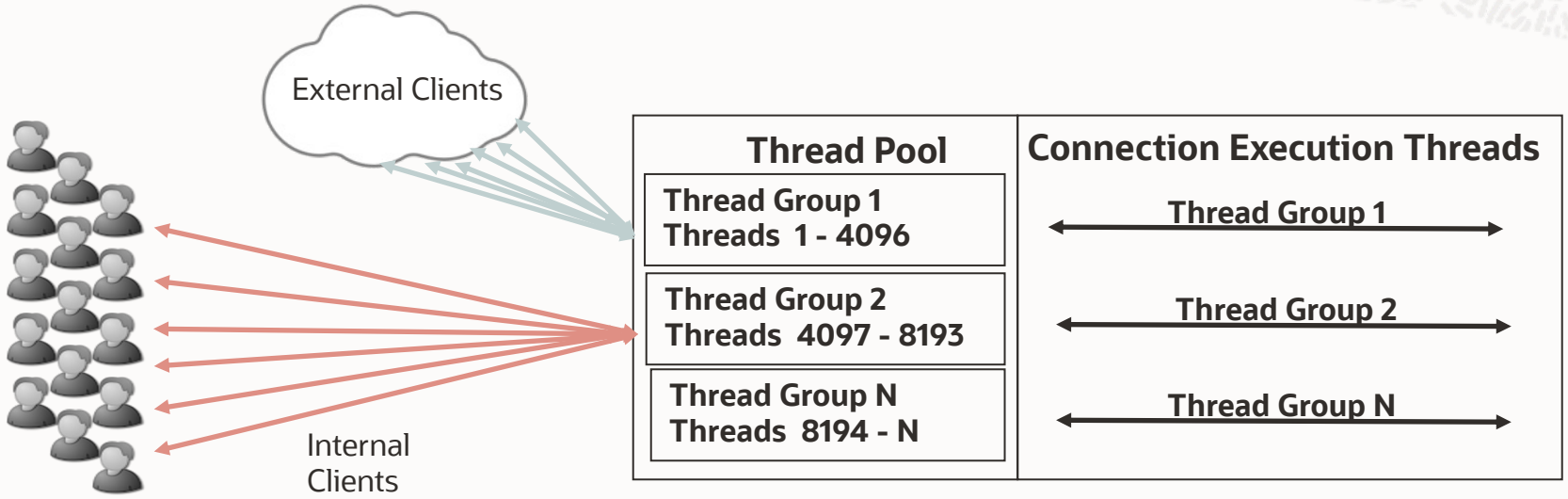
- In implementing a thread pool it is easy to cause a scalability problem by managing all threads in one pool.
- To avoid this issue we have divided threads and connections into thread groups. The aim of the implementation is to have one or zero queries executing per thread group at any time.
- Connections are bound to thread groups by round robin assignment at connect time.
- Each thread group maintains prioritized queues of connections awaiting execution.
- So problem solved using Divide-and-Conquer technique.

Thread Pool Solution: Minimize number of concurrent transactions

- To avoid scalability bottlenecks due to too many concurrent transactions it is imperative to limit the number of concurrent transactions.
- This is accomplished by prioritizing connections waiting to execute. Connections that have ongoing transactions are queued in a high priority queue and others are queued in the low priority connection.
- This means that low priority connections will wait until all high priority connections have completed.

MySQL Enterprise Thread Pool

Scales user connections



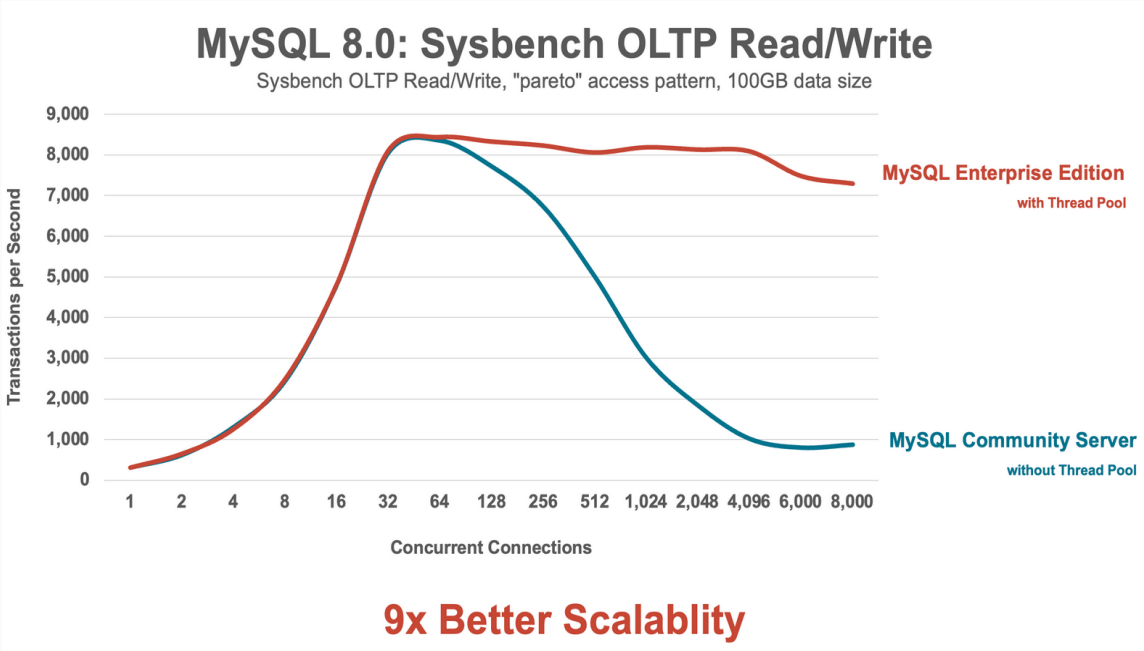
Provided by the thread pool plugin

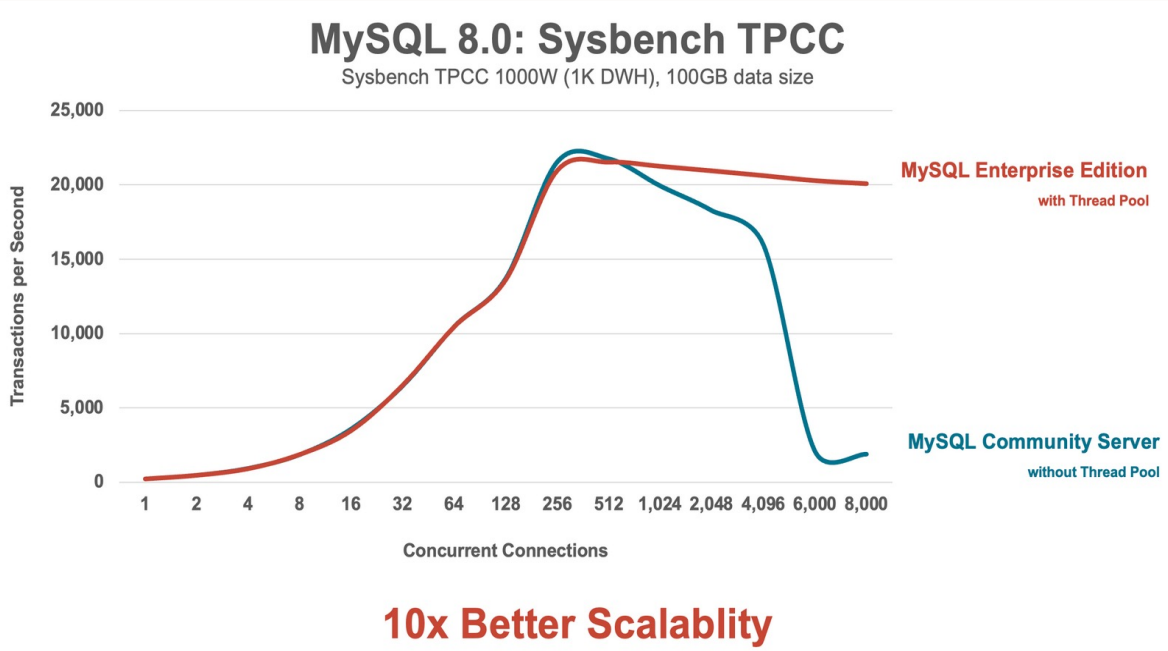
More intelligent model than the default

- Improves scalability as concurrent connections grow
- Threads are prioritized and statements queued

Performance Scalability with MySQL Enterprise Thread Pool

- To meet the sustained performance and scalability of ever increasing user
- A highly scalable thread-handling model designed to reduce overhead in managing client connections and statement execution threads





Server: 2 Sockets, 48cores, 96 threads Intel(R) Xeon(R) Platinum 8268 CPU @ 2.90GHz, RAM: 192GB
Storage: NVMe Optane 2 x 375GB, OS: OL7.9 UEK6, MySQL: v8.0.31 OpenSSL-1.1.1

Client: 2 Sockets, 44cores, 88 threads Intel(R) Xeon(R) CPU E5-2699 v4 @ 2.20GHz

Thread Pool vs. Connection Pool

- Given that a connection has been equal to a thread in MySQL, it is easy to confuse the Connection Pool with the Thread Pool.
- They are however solutions to different problems. A connection pool makes it possible to reuse connections and thus saving on having to execute connects to the MySQL Server.
- The thread pool works on the server side to limit the number of concurrently executing queries.
- Thus they can be used independently of each other.

Thread Pool: When to use?

- The most important variable to monitor is **`threads_running`**.
- This variables provides information about number of concurrent queries executed in MySQL Server.
- It is sufficient if this variable has spikes that put it above the optimal **`thread_pool_size`** for the type of queries used in the MySQL Server for the thread pool to be useful.
- Given that these spikes usually happen in overload situations, the thread pool is an important protection against troubles in overload situations.

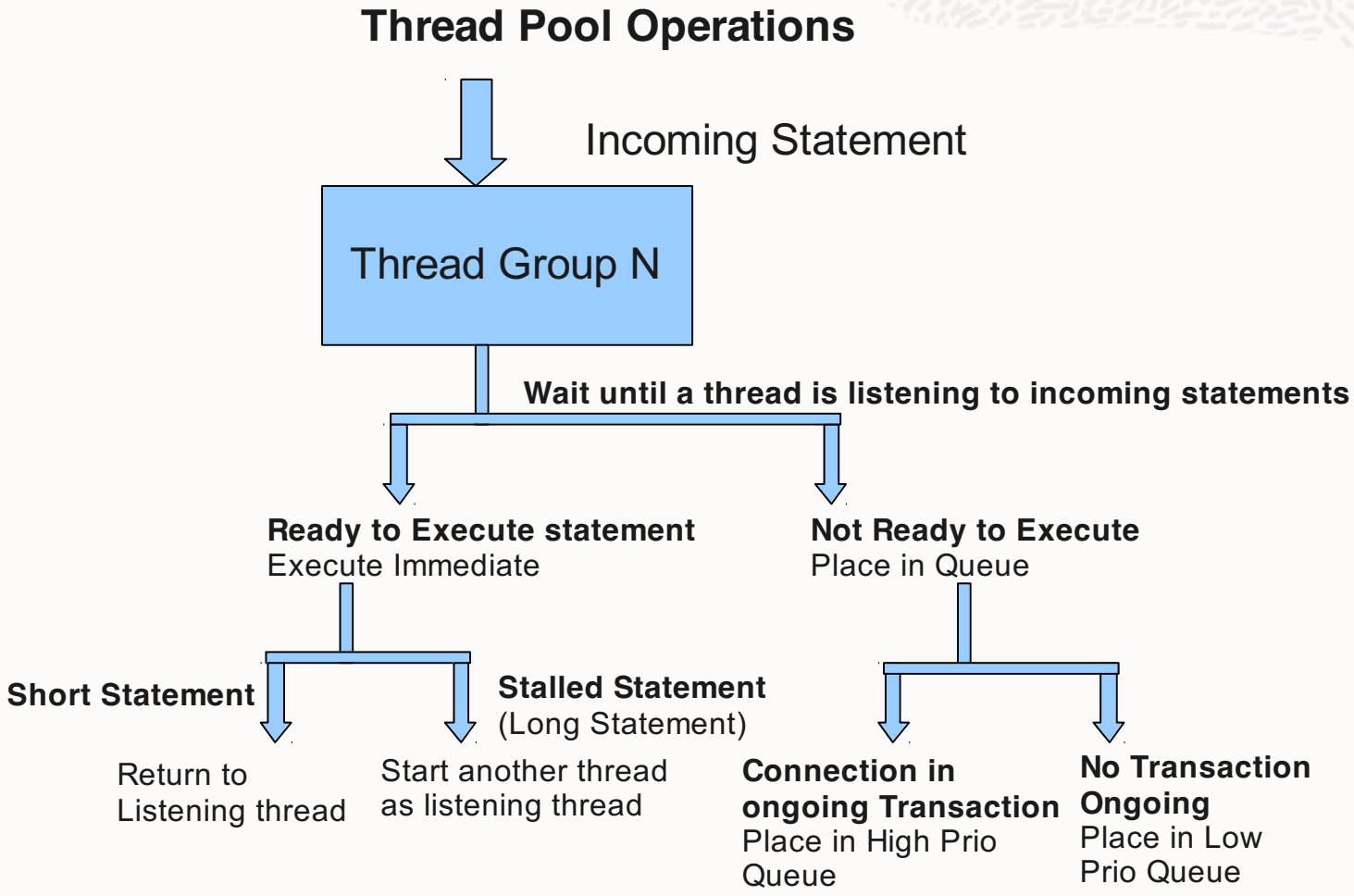
https://dev.mysql.com/doc/refman/8.0/en/server-status-variables.html#statvar_Threads_running
https://dev.mysql.com/doc/refman/8.0/en/server-system-variables.html#sysvar_thread_pool_size

Thread Pool: When to use (cont.)?

- If you are currently using `--innodb-thread-concurrency` to limit number of concurrent queries, then usage of the thread pool is very likely to be beneficial.
- Also if most of your queries are short queries then the thread pool is likely to provide benefits.
- Long queries are not negative for the thread pool since it will operate very similarly to the normal MySQL threading model for these queries. In these cases there will be fewer benefits to enabling the thread pool.

MySQL Enterprise Thread Pool

Flow Logic



Thread Pool: Operations

- thread_pool_dedicated_listeners: Dedicates a listener thread in each thread group to listen for incoming statements from connections assigned to the group.
- thread_pool_max_transactions_limit: The maximum number of transactions permitted by the thread pool plugin.
- thread_pool_query_threads_per_group: The number of query threads permitted in a thread group (the default is a single query thread).
- thread_pool_size: The number of thread groups in the thread pool. This is the most important parameter controlling thread pool performance.
- thread_pool_stall_limit: The time before an executing statement is considered to be stalled.
- thread_pool_transaction_delay: The delay period before starting a new transaction.
- thread_pool_algorithm: The concurrency algorithm to use for scheduling.
- thread_pool_high_priority_connection: How to schedule statement execution for a session.
- thread_pool_max_active_query_threads: How many active threads per group to permit.
- thread_pool_max_unused_threads: How many sleeping threads to permit.
- thread_pool_prio_kickup_timer: How long before the thread pool moves a statement awaiting execution from the low-priority queue to the high-priority queue.

Thread Pool: Optimal configuration

`--thread_pool_size` (Number of thread groups)

- InnoDB Read Workloads usually around 30-40
- InnoDB Write Workloads lower at around 12-30
- InnoDB Read/Write **Workloads at 16-36**

- Most important setting. Only set at server startup.
- Also is dependent on type of queries in workload, but default setting of 16 is usually a good starting point.
- Beneficial often to lock MySQL Server to same number of CPU threads as this parameter is set to.

Thread Pool: Optimal configuration (cont.)

`--thread_pool_query_threads_per_group` (Number of query thread in thread groups)

- Regular workloads – 2

- Also is dependent on type of queries in workload, but default setting of 1 is usually a good starting point.
- Beneficial often to lock MySQL Server to same number of CPU threads as this parameter is set to.

https://dev.mysql.com/doc/refman/8.0/en/server-system-variables.html#sysvar_thread_pool_query_threads_per_group

Thread Pool: Optimal configuration (cont.)

--thread_pool_stall_limit

- For benchmarks usually set to at least 1 second (=100).
- Check thread pool Information/Performance Schema tables to ensure not too many queries are stalled.
- Default setting is conservative to avoid impact of many long queries (=6, 60ms).
- Measured in 10ms segments. Setting of 6 = 60ms.
- Dynamic variable. Can be altered during runtime.
- Determine fraction of queries that are stalled (lower is better):
  - `SELECT SUM(STALLED_QUERIES_EXECUTED) / SUM(QUERIES_EXECUTED) FROM performance_schema.tp_thread_group_stats;`

Thread Pool: Optimal configuration (cont.)

`--thread_pool_prio_kickup_timer`

- For benchmarks usually set to at least 10 seconds (=10000).
- Set according to risk of livelock due to long-running queries.
- Default setting of 1 second should be OK for most cases, increasing this value is ok if the environment has only few and controlled long queries.

Thread Pool

Information Schema Tables

3 Tables:

`TP_THREAD_STATE`

- Shows current state of all threads controlled by thread pool

`TP_THREAD_GROUP_STATE`

- Shows current state of all thread groups in the thread pool

`TP_THREAD_GROUP_STATS`

- Shows statistics for each of the thread groups in the thread pool

Thread Pool Information Schema Tables:
TP_THREAD_STATE

Can view

- One row per thread created by plugin
- How long of time a query executed in thread (10ms segments)
- and if in a waiting state from where wait was initiated

https://dev.mysql.com/doc/refman/8.0/en/performance-schema-tp-thread-state-table.html

Thread Pool Information Schema Tables:
TP_THREAD_GROUP_STATE

Can view current values for example:

- Number of threads in thread group
- Number of connections in the queues
- Number of threads in various categories
- Number of connections per thread group
- Configuration settings
- For any listening thread
- Query with current longest wait in queues
- Number of active and stalled queries

https://dev.mysql.com/doc/refman/8.0/en/performance-schema-tp-thread-group-state-table.html

Thread Pool Information Schema Tables:
TP_THREAD_GROUP_STATS

Statistics per Thread Group. You can view:

- Thread group ID
- Number of connections started, closed
- Number of statements executed, queued
- Number of threads started
- Stats on consumer, reserve, waiter thread roles
- Stats on waits and locking

https://dev.mysql.com/doc/refman/8.0/en/performance-schema-tp-thread-group-stats-table.html

Thread Pool: Summary

Problem and how it is solved

Problem 1:

How to split threads into groups individually handled to avoid making the solution a problem in itself, aim is to manage one active thread per group?

Solution:

Connections are put into a thread group at connect time by round robin. Configurable amount of thread groups. This ensures that the thread pool itself isn't a scalability hog.

Thread Pool: Summary

Problem and how it is solved

Problem 2:

How to wait for execution of a query until the MySQL Server has sufficient CPU and memory resources to execute it?

Solution:

Each thread group tries to keep the number of executing queries to one or zero. If a query is already executing in the thread group, put connection in wait queue.

Thread Pool: Summary

Problem and how it is solved

Problem 3:

How to prioritize queries on connections that have an ongoing transaction?

Solution:

Put waiting connections in high priority queue when a transaction is already started on the connection.

Thread Pool: Summary

Problem and how it is solved

Problem 4:

How to avoid deadlocks when queries are stalled or execute for a long time?

Solution:

Allow another query to execute when the executing query in the thread group is declared as stalled (after a configurable time).

Benchmarking

Popular Generic Workloads

Sysbench

- Our recommendation
- Looks simple, but still the most complete test kit !
- OLTP, RO/RW, points on various RO and RW issues

TPC-C :

- DBT-2 / TPCC-like / HammerDB / Sysbench-TPCC

TPC-H / DBT-3

- DWH, RO, complex heavy queries, loved by Optimizer Team ;-)

dbSTRESS

- OLTP, RO/RW, several tables, points on RW and Sec.IDX issues

Why Sysbench is Favorite Choice

- Simple to install & use
- Covers most important MySQL Workloads
- Very flexible
  - LUA scripts
  - Easy to write your own tests
- Source available
  - https://github.com/akopytov/sysbench

Why Sysbench is Favorite Choice (cont.)

- Stress hardware
  - CPU
  - Fileio
  - Memory
  - Threads
- Compare Cloud nodes
  - Queries per $
- Assess feasibility of designs
  - Group Replication across broad networks
- Great for synthetic benchmarks

Installation & Usage (RPM)

Add the EPEL library:

- sudo yum -y install https://dl.fedoraproject.org/pub/epel/epel-release-latest-8.noarch.rpm

Install Sysbench:

- sudo yum –y install sysbench

Verify installation:

- sysbench –version

Useful Links:

- https://github.com/akopytov/sysbench
- https://www.mortensi.com/2021/06/how-to-install-and-use-sysbench/

Installation & Usage (RPM) cont.

Simple test of cpu:

- `sysbench cpu --cpu-max-prime=10000 run`

Simple test of MySQL:

- Create database schema
  - `mysql -uroot -pWelcome1! -e"CREATE DATABASE sysbench”`
- Prepare databases
  - `sysbench oltp_read_write --table-size=1000000 --db-driver=mysql --mysql-host=127.0.0.1 --mysql-db=sysbench --mysql-user=root --mysql-password=Welcome1!` **`Prepare`**
- Run sysbench
  - `sysbench oltp_read_write --threads=2 --report-interval=3 --histogram --time=50 --table-size=1000000 --db-driver=mysql --mysql-host=127.0.0.1 --mysql-db=sysbench --mysql-user=root --mysql-password=Welcome1!` **`Run`**
- Clean up
  - `sysbench oltp_read_write --db-driver=mysql --mysql-host=127.0.0.1 --mysql-db=sysbench --mysql-user=root --mysql-password=Welcome1!` **`cleanup`**

BMK Toolkit

- Toolkit created by Dimitri Kravtchuk (MySQL Performance Engineer)
- Designed to be an All-in-One toolkit.
  - Contains pre-built Sysbench binary statically compiled with the MySQL Client Library.
  - Fully contained tree pointed to by BMK_HOME environment variable
  - Includes TPCC & dbSTRESS
- Sample run:

```
$ bash /BMK/sb_exec/sb11-Prepare_50M_8tab-InnoDB.sh 32 # prepare data
$ for users in 1 2 4 8 16 32 64 128 256 512 1024
do
  # run OLTP_RW for 5min each load level..
  bash /BMK/sb_exec/sb11-OLTP_RW_50M_8tab-uniform-ps-trx.sh $users 300
  sleep 15
done
```

- Download:
  - http://dimitrik.free.fr/BMK-kit.tgz
- Docs
  - http://dimitrik.free.fr/blog/posts/mysql-perf-bmk-kit.html

Workshop Overview

- **Goals:**
  1. Create a OCI Compute server for hosting MySQL Enterprise Edition
  2. Install MySQL Enterprise Edition
  3. Overview & Setup Enterprise Scalability.

- **What this Workshop is not:**
  - In-depth tutorial on Oracle Cloud Infrastructure
  - MySQL Training Class

- **Lab:**
  - https://bit.ly/ThreadPool

MySQL Enterprise Edition - SECURITY

MySQL Enterprise TDE

- Data-at-Rest Encryption
- Key Management/Security

MySQL Enterprise Encryption

- Public/Private Key Cryptography
- Asymmetric Encryption

MySQL Enterprise Authentication

- External Authentication Modules
  - Microsoft AD, Linux PAMs, LDAP

MySQL Data Masking

MySQL Enterprise Firewall

- Block SQL Injection Attacks

MySQL Enterprise Audit

MySQL Enterprise Monitor

- Changes in Database Configurations, Users Permissions, Database Schema, Passwords

MySQL Enterprise Backup

- Securing Backups, AES 256 encryption

MySQL Enterprise Thread pool

- Attack Hardening

Thank you

Contacts:

Catherine Schrimsher
catherine.schrimsher@oracle.com

Eric Yanta
eric.yanta@oracle.com