MySQL @ Klaviyo Ops Crash Course

Author: Chris Miller

Claps: 66

Date: Nov 9, 2022

Iâ€™m a Lead SRE at Klaviyo. In the weeks leading up to Black Friday / Cyber Monday, my group takes over our weekly tech talk calendar to remind old Klaviyos â€” and teach new ones â€” about some of our baseline best practices. This is the content from our refresher session on MySQL at Klaviyo, with a few references to our internal tools and documentation left out.

Background

We run AWS Aurora MySQL, so the storage engine isnâ€™t as tightly coupled to compute as in vanilla MySQL. Aurora gives us faster failovers and replication, better durability by default, and scales to larger storage sizes. How this works is covered in this paper on Aurora from Amazon.

We use ProxySQL between clients and our higher volume MySQL clusters. This is primarily to pool connections across clients so we donâ€™t hit RDS limits. Many clients at Klaviyo hold open idle connections to a wide range of MySQL databases, so centralized connection pooling for some of them is essential. We recommend teams use ProxySQL if their database cluster sees connection counts consistently above 9,000 during our BFCM prep scaleout tests.

Common failure modes

If we see load-related issues during BFCM, they tend to be one of these:

- CPU exhaustion
- Freeable Memory exhaustion
- Low Buffer Cache Hit Ratio

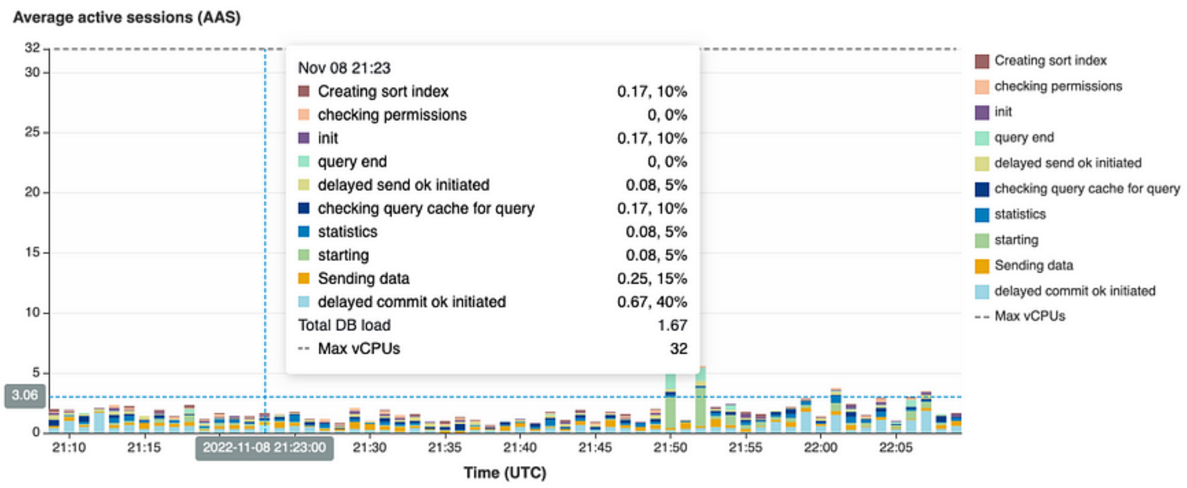
All of these failures can be caused by changes in query patterns. This could be an increased volume for existing queries or new queries that are poorly optimized (not using an index, for example).

CPU and memory exhaustion can be resolved by identifying the queries that are causing the issue, and mitigating them case by case. Another option is upsizing the writer / readers, but this requires failover and is disruptive. We aim to only do this well in advance of when needed and off peak hours.

How to troubleshoot workloads

AWS Cloudwatch Performance Insights

This is the first place to go when investigating any performance degradation issues with Aurora MySQL. Performance Insights generates graphs of a wide variety of performance indicators. It will also show you which SQL queries are taking the most time to process over time, as well as some info about why. Itâ€™s a good idea to bookmark the performance insights page for your database because when every second counts, you donâ€™t want to be navigating through the AWS console waiting for pages to load.

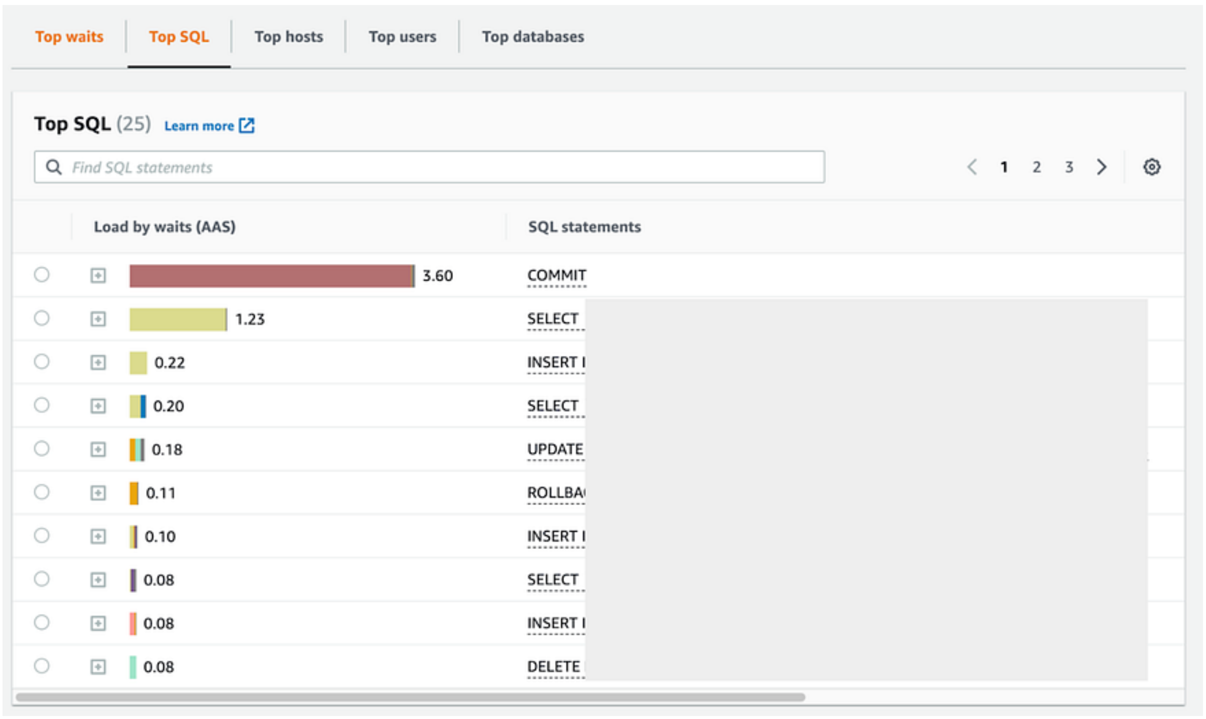


Nov 08 21:23	
🟫 Creating sort index	0.17, 10%
🟧 checking permissions	0, 0%
🟪 init	0.17, 10%
🟩 query end	0, 0%
🟨 delayed send ok initiated	0.08, 5%
🟦 checking query cache for query	0.17, 10%
🟦 statistics	0.08, 5%
🟩 starting	0.08, 5%
🟧 Sending data	0.25, 15%
🟦 delayed commit ok initiated	0.67, 40%
Total DB load	1.67
-- Max vCPUs	32

The Average active sessions graph shows which â€˜waitsâ€™ are placing the most load on a database over time.

Typically, performance insights makes it fairly clear how the performance of your queries have changed over time, and which queries contribute the most to load on the database. It also lets you drill down into the specific sub-operations performed during a query (the â€˜waitsâ€™) for each query pattern identified, so that you can make informed decisions about how best to mitigate the load placed on the database for that query.

For example, you might see that a particular query spends most of its time waiting on locks, which implies that contention for those locks is a problem, which you might mitigate by reducing parallelism in those queries or by sharding data sets to avoid contention.



Top SQL (25)

Load by waits (AAS)	SQL statements
3.60	COMMIT
1.23	SELECT [redacted]
0.22	INSERT I [redacted]
0.20	SELECT [redacted]
0.18	UPDATE [redacted]
0.11	ROLLBA [redacted]
0.10	INSERT I [redacted]
0.08	SELECT [redacted]
0.08	INSERT I [redacted]
0.08	DELETE [redacted]

The Performance Insights Top SQL list shows queries ranked by their load on the database.

The default Performance Insights view will rank the top queries made on a database instance over a given period of time by their total wait times, with break downs by wait type. This ranking alone often provides sufficient insight into which queries are generating the most CPU load, allowing you to make informed decisions about where to focus optimization or incident response efforts. Once you identify the heaviest queries, you can work with database clients to reduce the number or load of those queries on a case by case basis.

The power of these query performance measurements combined with the intuitive way that the UI allows engineers to explore them on the fly is why we recommend Performance Insights as a first stop for any database related troubleshooting. It can be incredibly powerful for incident response (especially given that 7 days of performance data history is included in the free tier, which is more than enough for heat of the moment incident response use cases).

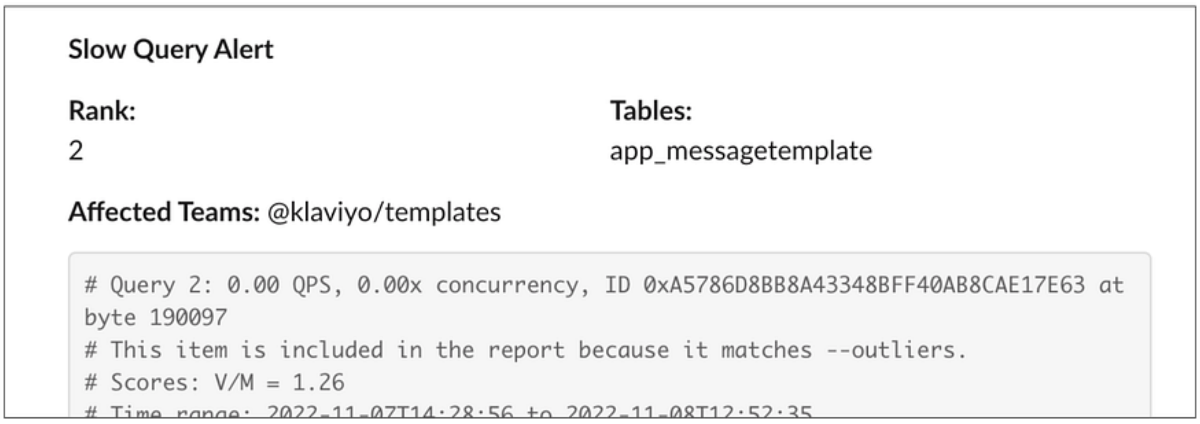
Slow Query Logs

You can download these from your instances, and process them with pt-query-digest to get another view into which queries are taking the longest. This gives slightly different information than Performance Insights.

Example:

```
# Query 1: 0.02 QPS, 0.07x concurrency, ID 0x64EF0EA1267300020...
# This item is included in the report because it matches --limit.
# Scores: V/M = 35.61
# Time range: 2022-11-06T14:00:52 to 2022-11-07T13:59:32
# Attribute    pct   total     min     max     avg     95%  stddev
# ============ === ======= ======= ======= ======= ======= =======
# Count         25    1434
# Exec time     95   6103s   872ms    324s      4s      9s     12s
# Lock time     97     32s    12ms   236ms    23ms    36ms    20ms
# Rows sent      0   1.05k       0      80    0.75    0.99    3.35
# Rows examine   0   1.82k       0     350    1.30    0.99   10.62
# Query size     1  84.02k      60      60      60      60       0
# String:
# Databases    app
# Time         2022-11-06... (1/0%), 2022-11-06... (1/0%)... 5 more
# Query_time distribution
#   1us
#  10us
# 100us
#   1ms
#  10ms
# 100ms  #
#    1s  ################################################################
#  10s+  ###
# Tables
#    SHOW TABLE STATUS FROM [redacted] LIKE '[redacted]'\G
#    SHOW CREATE TABLE `[redacted]`.`[redacted]`\G
#    SHOW TABLE STATUS FROM `[redacted]` LIKE '[redacted]'\G
#    SHOW CREATE TABLE `[redacted]`.`[redacted]`\G
[redacted SQL query]
```

At Klaviyo we also have a cron job that regularly pull slow query logs for many of our databases, and then notifies table/model owners according to codeowners in our #alerts-slow-queries slack channel. Example:


Slow Query Alert

Rank:
2

Tables:
app_messagetemplate

Affected Teams: @klaviyo/templates

```
# Query 2: 0.00 QPS, 0.00x concurrency, ID 0xA5786D8BB8A43348BFF40AB8CAE17E63 at
byte 190097
# This item is included in the report because it matches --outliers.
# Scores: V/M = 1.26
# Time range: 2022-11-07T14:28:56 to 2022-11-08T12:52:35
```

Example of PT Query Digest output that we post in teamsâ€™ slack channels.

SHOW PROCESSLIST

When a database is extremely degraded, for example when asked to execute a large number of long-running queries at once, a database ownerâ€™s best insights often come from the list of currently running queries.

You can use the MySQL console and check the process list with the SHOW PROCESSLIST command.

This will show you which queries are currently running and can be useful for identifying blocked queries that may have stacked up, with no chance of completing in a reasonable period of time. Often the play here is to try to identify where these queries are coming from and try to stop new ones from being executed.

Sometimes that isnâ€™t enough to return the database to satisfactory performance, since queries that clients have given up on will continue to run to completion on the server. In those emergency cases, we advise teams that they can KILL queries, but this should only be a last resort due to the potential for data inconsistency, and should not be taken lightly. You should never kill queries that you do not fully understand!

```
mysql> SHOW FULL PROCESSLIST\G
*************************** 1. row ***************************
     Id: 1
   User: system user
   Host:
     db: NULL
Command: Connect
   Time: 1030455
  State: Waiting for master to send event
   Info: NULL
*************************** 2. row ***************************
     Id: 2
   User: system user
   Host:
     db: NULL
Command: Connect
```

```
   Time: 1004
  State: Has read all relay log; waiting for the slave
         I/O thread to update it
   Info: NULL
```

Other tips

Timeouts

Take query timeouts seriously. At Klaviyo we use Sentry and treat query timeout exceptions as high priority. Queries that timeout at the client are still scheduled for execution on the server. So if theyâ€™re long lived they may stack up, even though their results have nowhere to go. The only way to pre-empt them is by killing them manually after consulting the SHOW PROCESSLIST output.

Instance sizing

Make sure that all instances in a cluster are the same size, otherwise replication can get wonky.

ProxySQL

Our general advice for ProxySQL is that itâ€™s not a silver bullet. Avoid using it unless you need it due to high connection counts. It comes with an operational burden that you may not need.

Metrics and thresholds

Here are some general guidelines we provide teams that own MySQL databases:

- **CPU Utilization** â€” if a database instanceâ€™s steady state (not peak) CPU utilization is consistently > 25% itâ€™s worth thinking about upsizing or otherwise optimizing workloads.
- **DatabaseConnections** â€" If this metric is consistently >> 9k look into slow queries, and consider using ProxySQL as a connection pool.
- **Buffer Cache Hit Ratio** â€" an underrated performance indicator. 99% is low, and can have a noticeable impact on query latency. It often indicates memory pressure. Remember it tends to be low on newer database instances while they populate their caches. Consider upsizing instances to provide more memory.

Useful links

- AWS Performance Insights docs
- Aurora MySQL Wait docs â€" use this lookup what the â€˜waitâ€™ types mean (e.g. io/aurora_respond_to_client)
- Percona Toolkit docs
- Percona Blog MySQL posts â€" lots of great content here