10 essential performance tips for MySQL

From workload profiling to the three rules of indexing, these expert insights are sure to make your MySQL servers scream

By Baron Schwartz

InfoWorld |
JUL 25, 2017 3:00 AM PDT

As with all relational databases, MySQL can prove to be a complicated beast, one that can crawl to a halt at a moment's notice, leaving your applications in the lurch and your business on the line.

The truth is, common mistakes underlie most MySQL performance problems. To ensure your MySQL server hums along at top speed, providing stable and consistent performance, it is important to eliminate these mistakes, which are often obscured by some subtlety in your workload or a configuration trap.

[Database slow? Improve the speed and scalability of your RDBMS with these 21 rules for faster SQL queries. | Keep up with the hottest topics in programming with InfoWorld's App Dev Report newsletter.]

Luckily, many MySQL performance issues turn out to have similar solutions, making troubleshooting and tuning MySQL a manageable task.

Here are 10 tips for getting great performance out of MySQL.

MySQL performance tip No. 1: Profile your workload

The best way to understand how your server spends its time is to profile the server's workload. By profiling your workload, you can expose the most expensive queries for further tuning. Here, time is the most important metric because when you issue a query against the server, you care very little about anything except how quickly it completes.

The best way to profile your workload is with a tool such as MySQL Enterprise Monitor's query analyzer or the pt-query-digest from the Percona Toolkit. These tools capture queries the server executes and return a table of tasks sorted by decreasing order of response time, instantly bubbling up the most expensive and time-consuming tasks to the top so that you can see where to focus your efforts.

Workload-profiling tools group similar queries together, allowing you to see the queries that are slow, as well as the queries that are fast but executed many times.

MySQL performance tip No. 2: Understand the four fundamental resources

To function, a database server needs four fundamental resources: CPU, memory, disk, and network. If any of these is weak, erratic, or overloaded, then the database server is very likely to perform poorly.

Understanding the fundamental resources is important in two particular areas: choosing hardware and troubleshooting problems.

When choosing hardware for MySQL, ensure good-performing components all around. Just as important, balance them reasonably well against each other. Often, organizations will select servers with fast CPUs and disks but that are starved for memory. In some cases, adding memory is a cheap way of increasing performance by orders of magnitude, especially on workloads that are disk-bound. This might seem counterintuitive, but in many cases disks are overutilized because there isn't enough memory to hold the server's working set of data.

Another good example of this balance pertains to CPUs. In most cases, MySQL will perform well with fast CPUs because each query runs in a single thread and can't be parallelized across CPUs.

When it comes to troubleshooting, check the performance and utilization of all four resources, with a careful eye toward determining whether they are performing poorly or are simply being asked to do too much work. This knowledge can help solve problems quickly.

MySQL performance tip No. 3: Don't use MySQL as a queue

Queues and queue-like access patterns can sneak into your application without your knowing it. For example, if you set the status of an item so that a particular worker process can claim it before acting on it, then you're unwittingly creating a queue. Marking emails as unsent, sending them, then marking them as sent is a common example.

Queues cause problems for two major reasons: They serialize your workload, preventing tasks from being done in parallel, and they often result in a table that contains work in process as well as historical data from jobs that were processed long ago. Both add latency to the application and load to MySQL.

MySQL performance tip No. 4: Filter results by cheapest first

A great way to optimize MySQL is to do cheap, imprecise work first, then the hard, precise work on the smaller, resulting set of data.

For example, suppose you're looking for something within a given radius of a geographical point. The first tool in many programmers' toolbox is the great-circle (Haversine) formula for computing distance along the surface of a sphere. The problem with this technique is that the formula requires a lot of trigonometric operations, which are very CPU-intensive. Great-circle calculations tend to run slowly and make the machine's CPU utilization skyrocket.

Before applying the great-circle formula, pare down your records to a small subset of the total, and trim the resulting set to a precise circle. A square that contains the circle (precisely or imprecisely) is an easy way to do this. That way, the world outside the square never gets hit with all those costly trig functions.

MySQL performance tip No. 5: Know the two scalability death traps

Scalability is not as vague as you may believe. In fact, there are precise mathematical definitions of scalability that are expressed as equations. These equations highlight why systems don't scale as well as they should.

Take the Universal Scalability Law, a definition that is handy in expressing and quantifying a system's scalability characteristics. It explains scaling problems in terms of two fundamental costs: serialization and crosstalk.

Parallel processes that must halt for something serialized to take place are inherently limited in their scalability. Likewise, if the parallel processes need to chat with each other all the time to coordinate their work, they limit each other.

Avoid serialization and crosstalk, and your application will scale much better. What does this translate into inside of MySQL? It varies, but some examples would be avoiding exclusive locks on rows. Queues, point No. 3 above, tend to scale poorly for this reason.

MySQL performance tip No. 6: Don't focus too much on configuration

DBAs tend to spend a huge amount of time tweaking configurations. The result is usually not a big improvement and can sometimes even be very damaging. I've seen a lot of "optimized" servers that crashed constantly, ran out of memory, and performed poorly when the workload got a little more intense.

The defaults that ship with MySQL are one-size-fits-none and badly outdated, but you don't need to configure everything. It's better to get the fundamentals right and change other settings only if needed. In most cases, you can get 95 percent of the server's peak performance by setting about 10 options correctly. The few situations where this doesn't apply are going to be edge cases unique to your circumstances.

In most cases, server "tuning" tools aren't recommended because they tend to give guidelines that don't make sense for specific cases. Some even have dangerous, inaccurate advice coded into them – such as cache hit ratios and memory consumption formulas. These were never right, and they have become even less correct as time has passed.

MySQL performance tip No. 7: Watch out for pagination queries

Applications that paginate tend to bring the server to its knees. In showing you a page of results, with a link to go to the next page, these applications typically group and sort in ways that can't use indexes, and they employ a `LIMIT` and `offset` that cause the server to do a lot of work generating, then discarding rows.

Optimizations can often be found in the user interface itself. Instead of showing the exact number of pages in the results and links to each page individually, you can just show a link to the next page. You can also prevent people from going to pages too far from the first page.

On the query side, instead of using `LIMIT` with `offset`, you can select one more row than you need, and when the user clicks the "next page" link, you can designate that final row as the starting point for the next set of results. For example, if the user viewed a page with rows 101 through 120, you would select row 121 as well; to render the next page, you'd query the server for rows greater than or equal to 121, limit 21.

MySQL performance tip No. 8: Save statistics eagerly, alert reluctantly

Monitoring and alerting are essential, but what happens to the typical monitoring system? It starts sending false positives, and system administrators set up email filtering rules to stop the noise. Soon your monitoring system is completely useless.

I like to think about monitoring in two ways: capturing metrics and alerting. It's very important to capture and save all the metrics you possibly can because you'll be glad to have them when you're trying to figure out what changed in the system. Someday, a strange problem will crop up, and you'll love the ability to point to a graph and show a change in the server's workload.

By contrast, there's a tendency to alert way too much. People often alert on things like the buffer hit ratio or the number of temporary tables created per second. The problem is that there is no good threshold for such a ratio. The right threshold is not only different from server to server, but from hour to hour as your workload changes.

As a result, alert sparingly and only on conditions that indicate a definite, actionable problem. A low buffer hit ratio isn't actionable, nor does it indicate a real issue, but a server that doesn't respond to a connection attempt is an actual problem that needs to be solved.

MySQL performance tip No. 9: Learn the three rules of indexing

Indexing is probably the most misunderstood topic in databases because there are so many ways to get confused about how indexes work and how the server uses them. It takes a lot of effort to really understand what's going on.

Indexes, when properly designed, serve three important purposes in a database server:

1. Indexes let the server find groups of adjacent rows instead of single rows. Many people think the purpose of an index is to find individual rows, but finding single rows leads to random disk operations, which is slow. It's much better to find groups of rows, all or most of which are interesting, than to find rows one at a time.
2. Indexes let the server avoid sorting by reading the rows in a desired order. Sorting is costly. Reading rows in the desired order is much faster.
3. Indexes let the server satisfy entire queries from the index alone, avoiding the need to access the table at all. This is variously known as a covering index or an index-only query.

If you can design your indexes and queries to exploit these three opportunities, you can make your queries several orders of magnitude faster.

MySQL performance tip No. 10: Leverage the expertise of your peers

Don't try to go it alone. If you're puzzling over a problem and doing what seems logical and sensible to you, that's great. This will work about 19 times out of 20. The other time, you'll go down a rabbit hole that will be very costly and time-consuming, precisely because the solution you're trying seems to make a lot of sense.

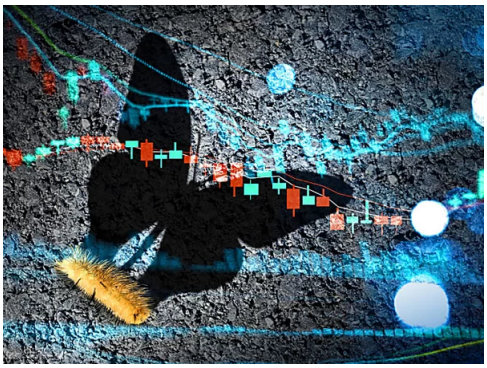
Build a network of MySQL-related resources – and this goes beyond toolsets and troubleshooting guides. There are some extremely knowledgeable people lurking on mailing lists, forums, Q&A websites, and so on. Conferences, trade shows, and local user group events provide valuable opportunities for gaining insights and building relationships with peers who can help you in a pinch.

For those looking for tools to complement these tips, you can check out the Percona Configuration Wizard for MySQL, Percona Query Advisor for MySQL, and Percona Monitoring Plugins. (Note: You'll need to create a Percona account to access those first two links. It's free.) The configuration wizard can help you generate a baseline my.cnf file for a new server that's superior to the sample files that ship with the server. The query advisor will analyze your SQL to help detect potentially bad patterns such as pagination queries (No. 7). Percona Monitoring Plugins are a set of monitoring and graphing plugins to help you save statistics eagerly and alert reluctantly (No. 8). All of these tools are freely available.

Baron Schwartz is a database performance and scalability expert, the lead author of "High Performance MySQL, 3rd Edition" (2012, O'Reilly), and the founder of VividCortex. He was previously chief performance architect at Percona.

Follow

Why MongoDB is 'fundamentally better' for developers



Fable F# to JavaScript compiler improves usability, speed

How IT priorities are shifting during the COVID-19 crisis

Anti-adversarial machine learning defenses start to take

Microsoft adds a new Linux: CBL-Mariner

Oracle adds analytics to MySQL in the cloud

The 24 highest paying developer roles in 2020

4 nonprofits teaching disadvantaged students to code