PRACTICAL

MySQL Performance Optimization

A hands-on, business-case-driven guide to understanding MySQL query parameter tuning and database performance optimization.

PERCONA

With the increasing importance of applications

and networks in both business and personal interconnections, performance has become one of the key metrics of successful communication. Optimizing performance is key to maintaining customers, fostering relationships, and growing business endeavors.

A central component to applications in any business system is the database, how applications query the database, and how the database responds to requests. MySQL® is arguably one of the most popular ways of accessing database information. There are many methods to configuring MySQL that can help ensure your database responds to queries quickly and with a minimum amount of application performance degradation.

Percona is the only company that delivers enterprise-class software, support, consulting, and managed services solutions for both MySQL and MongoDB® across traditional and cloud-based platforms that maximize application performance while streamlining database efficiencies. Our global 24x7x365 consulting team has worked with over 3,000 clients worldwide, including the largest companies on the Internet, who use MySQL, Percona Server, Amazon® RDS for MySQL, MariaDB® and MongoDB.

This book, co-written by Peter Zaitsev (CEO and co-founder of Percona) and Alex Rubin (Percona consultant, who has worked with MySQL since 2000 as DBA and Application Developer) provides practical hands-on technical expertise to understanding how tuning MySQL query parameters can optimize your database performance and ensure that its performance improves application response times. It also will provide a thorough grounding in the context surrounding what the business case is for optimizing performance, and how to view performance as a function of the whole system (of which the database is one part).

THIS BOOK IS PRESENTED WITH THE FOLLOWING SECTIONS:

- **Section 1** *(this section): Application Performance and User Perception*
- **Section 2:** *Why is Your Database Performance Poor?*
- **Section 3:** *MySQL Configuration*

For more information on Percona, and Percona's software and services, visit us at **www.percona.com**.

ABOUT THE AUTHORS

Peter Zaitsev, CEO

Peter Zaitsev is CEO and co-founder of Percona. A serial entrepreneur, Peter enjoys mixing business leadership with hands-on technical expertise. Previously he was an early employee at MySQL AB, one of the largest open source companies in the world, which was acquired by Sun Microsystems in 2008. Prior to joining MySQL AB, Peter was CTO at SpyLOG, which provided statistical information on website traffic.

Peter is the co-author of the popular book, High Performance MySQL. He has a Master's in Computer Science from Lomonosov Moscow State University and is one of the award-winning leaders of the world MySQL community. Peter contributes regularly to the Percona Performance Blog and speaks frequently at technical and business conferences including the Percona Live series, SouthEast LinuxFest, All Things Open, DataOps LATAM, Silicon Valley Open Doors, HighLoad++ and many more.

Alexander Rubin, Principal Consultant

Alexander joined Percona in 2013. Alexander worked with MySQL since 2000 as a DBA and Application Developer. Before joining Percona he was doing MySQL consulting as a principal consultant for over 7 years (started with MySQL AB in 2006, then Sun Microsystems and then Oracle). He helped many customers design large, scalable and highly available MySQL systems and optimize MySQL performance. Alexander also helped customers design big data stores with Apache Hadoop and related technologies.

Introduction:

"Software is eating the world."

INTRODUCTION

There is a common phrase you're probably familiar with: "software is eating the world." It was coined by Marc Andreessen in an essay on the implications of a software-driven society. It's truer today than it was then. Indeed, applications impact and infiltrate more areas of our life every day. There isn't an aspect of our lives that doesn't have an application: we're using applications to share stories with our friends, share our kitten photos, and even share cabs.

As applications become a larger and larger part of our life, our expectations for those applications are also growing larger. We want applications to be always up, bug free, easy to use, secure, and well performing. In this book, we're going to talk about this last (but not least) application quality: performance. This book will cover the details of what application performance is, how we define it, and how we measure it.

Once we have defined application performance and examined how to properly measure it, we will discuss approaches to getting performance to the correct level, along with the most practical ways of doing it (without overspending, wasting developer resources, or creating impossible-to-maintain applications).

It is worth noting that performance goes hand in hand with availability. Availability is often viewed as much more important than performance, and managed much more tightly in many organizations. If your application responds so slowly that the user loses patience and quits waiting for a reply, the result is viewed as application downtime.

It goes without saying that users love systems that perform well, and are always available. Over the years, we've amassed a great deal of research on how performance impacts user behavior, revenues, and business success. Here are a few specific examples:

- ***In 2006, Amazon reported 100ms reduction in page speed that caused 1% increase in revenue***
- ***In 2008, Google reported an increase of page load time from 400ms to 900ms that decreased traffic and ad revenues by 20%***
- ***In 2009, Akamai reported that 40% of users will abandon a website if it takes more than 3 seconds to load***

These statistics obviously demonstrate the link between ensuring great performance and your business' success. If you're not working on improving performance, certainly your competitors are.

Even though it is obvious that ensuring system performance and availability is very important for business' success, many members of developer and operations teams don't understand performance very well. This lack of understanding can either cause application performance problems, or prevent resolving them in a timely manner. For example, developers often test their code using a small database, when the application is meant to interact with a large one. This can mask scalability and performance issues. Understanding how performance affects business decisions and business outcomes is an excellent tool at any level of the application management: it is good for career advancement as well as good for the company.

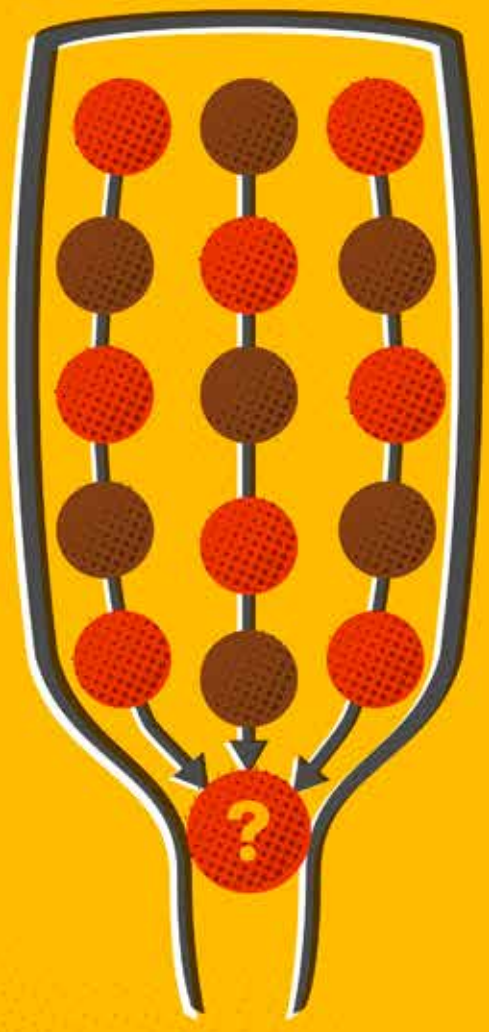
Viewing application performance from the 30,000-foot level is really looking at how quickly applications respond to user requests (where the "user" can also be other systems) in a given circumstance. Applications typically contain many components, and most of those components will contribute to response time.

What limits application performance depends primarily on the application. In his 2004 book High Performance Web Site, Steve Souders discovered 80-90% of response time comes from the application front end. When investigating performance, the front end is the first place to investigate (if the application backend is well optimized). On the other hand, at Percona we often run into applications where the backend is a problem--and more specifically, the database.

In addition to being a common bottleneck, a database is often harder to scale compared to an application's front end. Web servers, application servers, and the client-side workload are often easy to scale linearly with the growing number of users. The database, however, does not linearly scale so easily. If you don't maintain and maximize database performance, you risk it becoming a performance bottleneck.

Percona's experience and expertise with troubleshooting and optimizing database performance, and more specifically MySQL®, is the subject of this book. However, even if you're using some other database technology (such as NoSQL, MongoDB®, or others) you should find many of the concepts and ideas expressed in this book very helpful.

Application Performance



APPLICATION PERFORMANCE

In many of my conference presentations, I like to throw up a slide that states: "Nobody cares about MySQL performance." (Depending on the audience I might replace "nobody cares" with something stronger!) It definitely gets people's attention. When they start to argue about how wrong I am, I follow it with this slide: "Users care about application performance."

In information technology systems, performance is defined as the amount of useful work a system accomplishes in relation to time and other resources, such as memory, disk IO or CPU usage. Performance is typically measured either by throughput (the amount of work done per period of time) or response time (how quickly system responds to the user request), which in the context of database systems refers to the SQL statement or other API call.

It is crucial to understand this subtle difference in thinking. While it is true that many times when we improved application performance, we did so by optimizing MySQL, in other instances that was not the case.

As an example, when the Facebook Platform for developers was first released, its early users came to Percona to help fix their MySQL database performance. After investigating the problem, it turns out it was API calls to the platform that were taking a long time and causing application problems (not MySQL).

With another customer, we looked at their database and didn't find any significant load. When we asked them why they thought it was the database, their reply was, "Because it is always the database!"

This attitude is demonstrative of how many developers look at database performance.

So while the database may well be the bottleneck for application performance, it is much better to start with understanding application performance in general, it's various causes from the front to back end, and to drill down from there. As you do, you will come to the database, if it is indeed the cause of the problem.

When it comes to applications, a bottleneck is any component system, device, resource, or application that impacts the performance of the whole system. This could include:

- *Software programming*
- *Caching issues*
- *CPU issues*
- *Operating System deficits*
- *Network fluctuations*
- *Memory issues*
- *Database design*

There is another important reason to look at the entire application when investigating performance: even if the performance problem is caused by the database (the "bottleneck" in this case), it doesn't mean the solution lies with the database as well. In many cases you can resolve database issues by reducing the amount of load it has to handle, instead of just making it faster. This can be done, as an example, by enabling some form of application caching.

As much as users, your boss cares about application performance—or your boss's boss. The higher up you go in the management chain, the less interest there is in understanding the technical reasons that are causing application performance to degrade. Rather, they mostly care about ensuring that applications work quickly overall (and thus customers are happy). Embracing this aspect of your job can be helpful when review time rolls around! This is known as response time.

In any system, response time is the amount of perceived time it takes for the user to get feedback from entering input. "Good" response time is relative to the operation--a front page needs to respond quickly, whereas a search query is not as time-sensitive. Many (or all) aspects of a system can contribute to response time.

Understanding response time from the perspective of a given user's interactions is a great way to look at application performance problems. It points to where exactly the response time comes from, where you need to focus, and how much performance optimization is potentially possible. For example, if a one second response time contains 800ms that comes from the database, we can reduce total response time by almost 80% (in a best case scenario) with database optimization. If only 200ms is the database's responsibility, pursuing other optimization avenues may be your first priority (or at least in parallel with database optimization).

This important insight comes from proper "application instrumentation," which allows you to understand how a given request or type of requests affect response time. Many developers create and implement their own simple instrumentation, which can grow into very powerful in-house performance management tools. Many leading enterprise and web companies use such in-house solutions. Or you could also consider using tools like NewRelic and AppDynamics, which provide easy-to-use instrumentation and can easily analyze data.

A common ad-hoc approach to diagnosing performance issues is using special code as instrumentation. The special code could include routines for measuring and logging timing for certain application functions. This allows you to measure how much time the suspect code takes to execute--demonstrating where the program might be contributing to response time issues.

Finally, using solutions like Dtrace or SystemTap can be very powerful, and allow you to add the measurement data point to the code running the application without recompiling the application.

Since instrumentation support depends significantly on the programming language, platform, and development platform in use, we'll leave a more detailed instrumentation discussion to other sources.

If it isn't possible to instrument the application, we often have to take the opposite approach by looking at the possible bottleneck suspects (often the database in our case). This can be done by enabling the database log file in the development environment check what the application is doing. For example, using the logs

to observe what queries the application makes, their number, and their response time, and then measuring that against the full application response time allows us to gauge what portion of the response time is the database's responsibility. This method is easiest to apply in a development environment where there is only one person using the system. (It can also be done using the production environment, but it is more complicated and depends a great deal on the environment parameters.)

Whether the database is at fault, or the problem lies elsewhere in the system, it is very important that all the development and operational teams maintain a good working relationship. Too often we witness the blame game: trying to throw the problem over the fence rather than focusing on solving the problem at hand. It is common to see the development team hold the attitude that "...we should be able to run any queries we want, as often as we want, and it is up to the operations team to make sure it works!" Conversely, the operations team often embody the attitude that ". . . if only developers wouldn't run such complicated queries, the database would perform well!"

The reality is both of those approaches are selfish, and defeat finding a solution: you can't expect a database to run all the queries you throw at it, but some applications require complex database queries. To reach optimal results, the development and operations teams need to work together during both the design phase and during the troubleshooting phase, and be ready to give concessions.

Understanding Performance

UNDERSTANDING PERFORMANCE

In many cases, performance is in the eye of the beholder. Users perceive how systems behave through their expectations, and compare the performance against similar systems to make an assessment about whether performance is outstanding or inadequate. As the world changes, so does the user's attitude toward performance. In the late 1990s, I remember having Internet access through a 14400-baud modem. Waiting for minutes for a page to load was common. In today's world, few people have the patience to wait more than few seconds for a page to load. An application appears broken if its interactive elements do not respond within fractions of a second.

What is considered to be adequate performance is not only defined by user expectations, but also by specific use cases. Financial markets that employ stock trading are known to be very sensitive to response time, to the point where the physical location of the trading server in proximity to the stock exchange matters (due to communication delays over the physical media).

A system or application will often need to support multiple user interactions. For example, a typical ecommerce site has a search function, product detail page, and checkout function—and all of them have different response time criteria. "Landing" pages often have one of the lowest response time requirements, as it is crucial for making a good first impression. A search function in many cases does not have as strict a response time criteria, as searches can be computationally expensive and users expect them to take time. A checkout function also isn't nearly as time-critical, as communicating with external payment gateways takes some effort. If we look at the internal "back office" functions, such as reports of items sold, these often have even more relaxed criteria.

Users and Performance

Just as all application functions are not created equal, all users are also not created equal. There is a good chance that many performance problems will correspond to specific users or data objects. Often some operations teams get so familiar with certain user issues that they immediately recognize them, and you can hear team members say something like "Ah, user 1256 is trying to browse his one hundred thousand photos again!"

When problems are specific to certain users or certain data objects, this typically means they are outliers to some extent: holding much more data than the average user, having different cardinality, producing a large amount of requests, or have those requests being unusually slow and expensive.

You should not ignore these outliers for two reasons. The first is because your problem children are often your most valuable users and most active champions of your product. In the stock photography website example above, the user has one hundred thousand pictures in their portfolio. The user is one of their top sellers and is much more important to the business than the average user. The second reason is such users not only can have a poor performance experience themselves, but in many cases they can take up resources and impact the performance experience for other users as well.

For example, you may have heard about Facebook's "Justin Bieber" problem. Justin Bieber had more than 72 million likes on his Facebook page at the point of this writing (and counting). I would think having a page on Facebook is good both for Mr. Bieber and Facebook, but it does require some special engineering to support that level of user.

Investing engineering resources to support extreme use cases is one option. Another might be to simply impose some limitations. For example, LiveJournal for many years had a limit of 750 "friends" because they could not support more than that while maintaining consistently high performance. An even more prominent example is Google, which does not return more than 1000 search results for any query. Having clear limits in the software is often a much better choice than having users getting themselves in situations where the application starts to perform poorly (and builds up user frustration).

A final option is firing abusive users. Many low-cost shared hosting providers with "unlimited" space and traffic take this approach. The user one thousand times more active than the average user is quite likely to cost too much to support, and will be asked to leave (or "upgrade") to higher-end dedicated hosting.

Timing is Everything

Performance can also be impacted by time. First, there is the typical variation in the system load between daytime and nighttime (even global systems tend to have significant variance) where performance during peak times might suffer if the system gets overloaded. Second, it is possible for internal system process cycles to impact system performance. This might be caused by running automated scripts for hourly reporting, nightly backup processes, or monthly billing applications. Such scripts can cause significant impact to system performance, to the point where some systems are known to have higher user-facing performance problems during nighttime when the interactive users load is ostensibly the lowest. The performance during these times is impacted by the maintenance scripts that often get scheduled during "down" time.

That's So Random

Finally, the universe is a random place. There is a lot of strange stuff that goes on that can affect application performance. Perhaps a Google Bot (if you're lucky) or Spam Crawl Bot (if you're not) decided to give your website an intensive crawl. Maybe your application just became famous by being mentioned on TV (or even worse, Justin Bieber just wrote about it on his Facebook page). Maybe your hosting provider is having some issues, or, especially in cloud and virtualized environments, the neighbors sharing your infrastructure started to use a lot of resources. Figuring out what the universe decided to lay at your doorstep is often difficult. Sometimes performance problems come and go and never happen again. Their exact cause remains unknown.

How to Look at Response Time

Up until now, we have been discussing system response times in the context of individual transactions by specific users. In the large-scale systems, there will be millions or billions of users—analyzing every one of them isn't possible. So how can we look at all this data?

First off, the wrong way to look at response time is through the "average." There is a saying you may have heard: "I once knew a man who drowned trying to cross a stream that was three feet deep—on average." (You will hear many similar anecdotes in system performance expert circles.) The point is that looking at the average response time in most cases is a bad way to address the problem.

At the same time, looking at the maximum response time is also rarely a good idea (unless you're working with hard, real-time systems). The complexity of modern systems guarantees that some of the billions of transactions will be much slower than you would like.

Instead, the best option is to look at the percentile response time. For example, the 99th percentile shows a response time that 99% of all requests were able to match or beat. If the 99th percentile for a landing page opening was 5ms, then the response time of 99% of people going to that landing page was 5ms or better. Applications that are very focused on performance and have a large volume of requests—Amazon.com for example—might set even higher criteria, such as the 99.9% percentile. This would allow only 1 out of every 1000 requests to be an outlier.

As previously discussed, since response times can vary significantly due to time of day, user interaction, server, or database, it often makes sense to set a different percentile response time for each variable and watch for outliers based on the modifying factors. Using the stock photo website, for example, we might look at the 99% response time for user interaction to browse photos every five minutes, but use the 95% response time for every user on daily basis.

You might ask why the difference in the percentile figures and the time of the activity. The answer is sample size. In order to compute reliable percentile numbers you have to have decent sample sizes. You can't really compute a 99th percentile response time if there are only two requests in the sample that match the criteria. A good rule of thumb (OK, my rule of thumb!) Is that you should allow for at least 10 outliers when you compute percentile figures. For

example, if we're looking to compute a 99% response time for a given transaction every 5 minutes, I need to ensure I'm observing at least 1000 events during this time (10 being one percent of 1000).

Since response time is what is readily apparent to users—and the focus of their concern—it is obviously a key metric to track. But what other readily available metrics should we be concerned about? Should we be concerned with rates such as the number of requests per second (throughput) or utilization (as percent of CPU usage)?

While these metrics can often be very helpful benchmarks for capacity planning and performance analyses, they should be seen as a secondary metrics—mainly because they are not what your user cares about. In fact, the relationship between such metrics and performance can be very complicated.

Another thing to consider is that in real world systems, response time impacts the inflow of requests in various ways. Sometimes users faced with a slowly responding system will start clicking reload, increasing the inflow of complicated requests and overloading the system. We often see this as many orphaned queries on the database level that are sometimes left running for hours. At the same time, users also tend to abandon systems that perform very poorly, reducing inflow of requests and letting them recover.

In a different example, it is entirely possible for a system to have 100% CPU usage when handling a large load without seriously increasing response time. Chances are the CPU load is attributable to low priority processes that yielded when resources were needed to handle more important functions.

The "requests per second" is often a great way to gauge the system load. The system load often affects the response time. This same metric is also helpful for benchmarking and capacity planning. For example, if I changed the MySQL configuration and I see the throughput reduced by 50%, it definitely shows I should rethink the configuration adjustment—without employing an in-depth response time analyses.

Another class of performance-related metrics that can be helpful is "resource use metrics," such as number of IOs, CPU time used, memory usage, and network bandwidth. These all correspond to physical resource usage, and it is important to monitor these resources, as there is only so much to go around. If a system runs out of these resources, the system behavior is likely to change and performance will be drastically impacted.

For example, running out of available CPU time causes queuing in the run queue and contributes to increasing the response time. Running out of memory can cause system swapping, drastically slowing down the system.

Going forward, we will focus a great deal on the response time, contributors to response time, and the how this metric affects the end user. However we will also discuss other metrics related to performance, such as the number of requests per second, amount of memory used, or number of input/outputs per second (IOPS) performed when appropriate.

Another important concept when working on performance problems is "causality versus correlation." While observing a system exhibiting slow user interaction response times, you might wonder which user interactions are hindering system performance because they take a lot of resources, and which are slow because resources

have been exhausted. Sometimes it is very clear which one is the issue. In other cases, however, the answer might not be so simple.

For example, you might have a system able to handle 10 concurrent requests in parallel efficiently. However, 100 requests at the same time can overload the system due to not having enough resources to efficiently process them. In such cases it is not specific user interaction that caused the problem, but rather the fact so many were attempted at the same time.

Another example of causality versus correlation (a fancy way of saying "finding the true cause") is seeing the number of requests on the database go up when application response time decreases dramatically. You could claim that a specific number of requests caused the application to respond slower. In reality, however, it could be issues with the caching infrastructure that are causing a high cache miss rate—generating an increase in the number of queries to the database and application slowness.

It's not always clear what is causing a performance issue. It's not always easy to distinguish the cause of a performance issue from something that is coincidentally happening at the same time. Deep investigation may be needed to find a performance issue's root cause (or causes). If you keep response time as your focus, however, you have a much better chance at figuring out which is which. We will talk more about this in one of the next chapters.

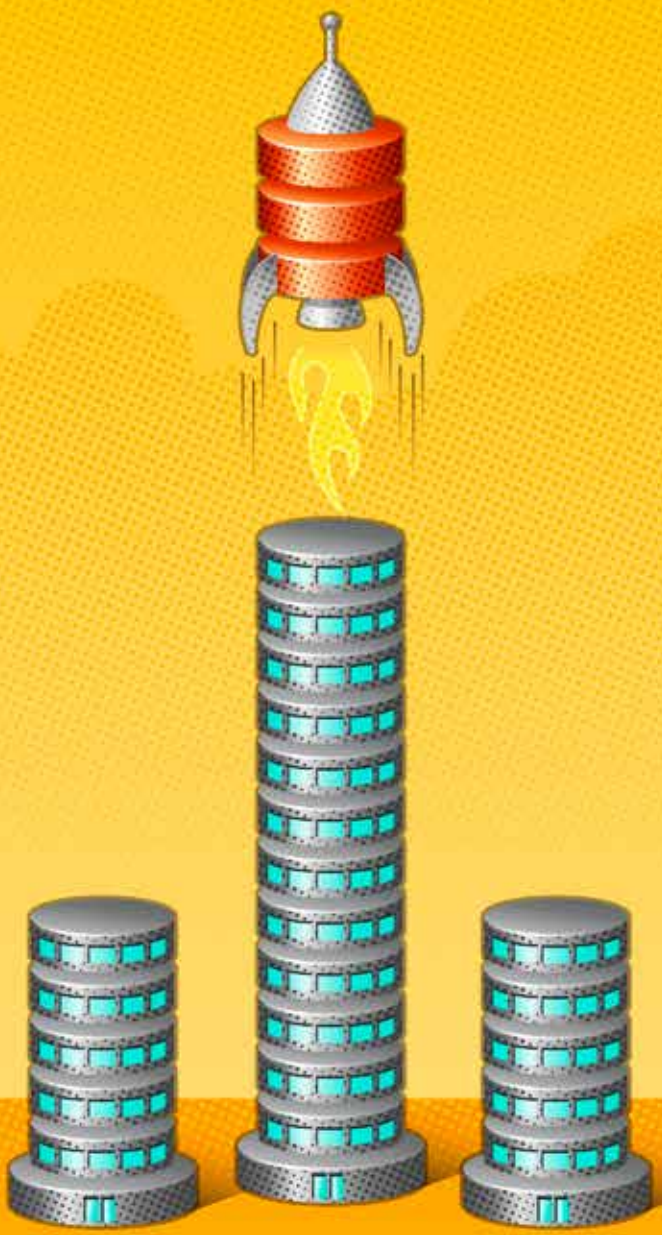
Finally, it is important to realize that performance optimization is a never-ending story. There is always a way to further improve system performance and reduce resource usage. Typically, however, each additional performance gain comes with higher and higher cost. The ends don't justify the means at a certain point. For

example, you can't get a round trip from the USA to India in 1ms at any cost (unless you figure out how to break the speed of light!).

Since you can always invest more into performance pursuits, and because performance optimization has a law of diminishing returns, you need to understand how much performance you really need. What response times are acceptable for users? How efficient does the system need to be? How expensive and complicated would it be to reach these goals?

You should always think about the alternatives uses of the time and costs of optimizing beyond the return on investment (ROI). There is a point where you should stop specific optimization efforts and focus on other options: optimizing different parts of the system, building new features or improving quality. We will talk more about more specific prioritization methods as we go along.

Business Needs



BUSINESS NEEDS

Application users look at performance very selfishly: if an application responds quickly to their requests, they are happy. Keeping users happy, at least most of them most of the time, is typically an important business goal—however it is not the only goal.

When working on system performance optimization, it is important to understand what business needs exist that justify or prevent implementing your solutions.

Consultants are very familiar with the concept "true, but useless, advice." The phrase means examining a problem, offering a valid solution that applies, and then realizing it can't be implemented due to current business realities or office politics. An example of this effect might be something like suggesting a department using Microsoft move to Linux, or move to a physical hardware environment, right after the CIO spent fifty million dollars implementing a completely virtualized enterprise environment.

The following are a few things businesses typically care about in addition to getting the best performance possible.

Agility

Agility is perhaps the most common business requirement that impacts application performance. Many businesses want software development to be agile—able to move fast and deliver new features to the market quickly. This often means the shortest route to implementation will be used, versus planning for high performance. In many cases performance design and capacity

planning won't even be addressed at the outset, but instead tackled when the system starts to experience performance problems.

You will often see a great deal of resistance to performance optimizations that impact development agility, complicate the development process, or slow it down. This often means using ideas that aren't the most direct route to solving the problem. Resistance to a more optimal solution could mean implementing queries manually instead of using an ORM framework in critical performance cases, implementing caching, or using replication or sharding.

Looking Ahead/Supporting Growth

Performance optimization is generally done for a specific scale. If that scale changes—more data, users, etc.—to say ten times the current architecture, then the current performance optimization might not be sufficient (or might even be detrimental by increasing architectural complexity). When addressing performance issues, it necessary to understand what direction the application or system is heading, and how the change may affect performance optimization requirements.

Resource Efficiency

Many people think that a high performance system is the same as an efficient system. It isn't. For example, the fastest sports car isn't the most fuel-efficient. This is also true when scaling systems. Highly scalable systems typically need to be distributed, and distributed systems tend to be more complicated and less resource efficient than systems that aren't. Knowing if your main goal is optimization or resource efficiency is important, as it will affect many of the decisions you make.

In computing, resources are any physical or virtual system component that has limited availability. Resources included things like CPU, memory, I/O operations, network bandwidth, number of file descriptors process can use, etc.

It is also important to prioritize what resources are optimized first, as not all resources are created equal. Saving one often means spending more of another. For example, saving on disk space often employs compression, which causes more CPU power usage.

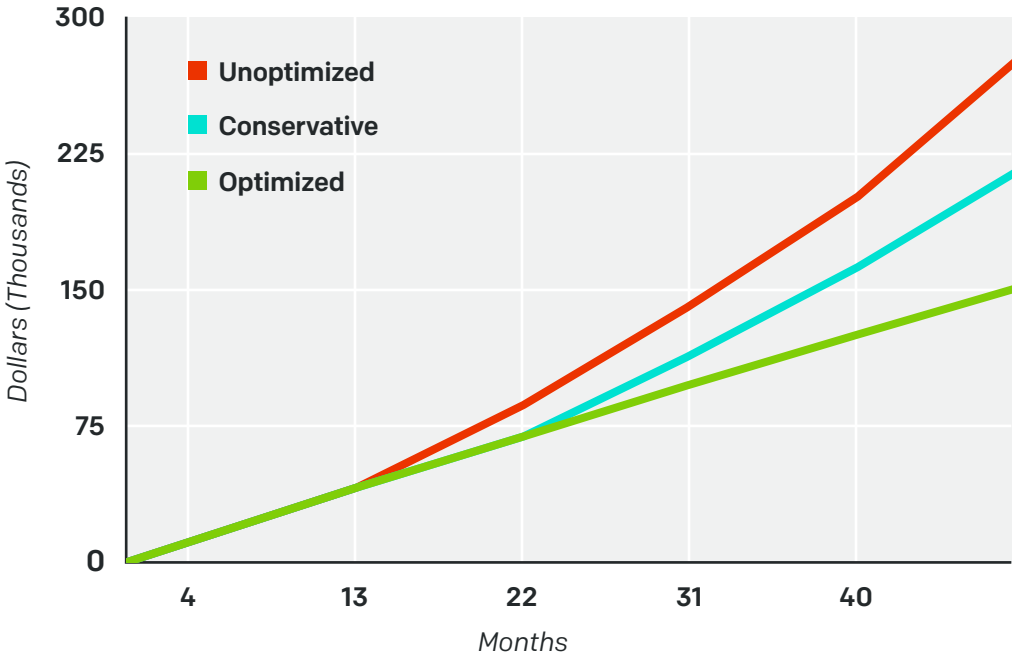
Costs

Costs are separate from resource efficiency because costs can be driven by other factors besides resources. For example, many companies are looking to be environmentally conscious these days, while others might simply have space limits in their data center.

Businesses watch costs, and performance optimization can save money by either directly reducing resource usage or by avoiding additional future investments.

As of November 2015, a single master-slave configuration, with a database size of 500GB, with 500 IOPS on the master, hosted on Amazon's AWS, can cost around $3,200 per month, or roughly $38,400 a year in hosting fees. With conservative growth, this could balloon up to around $4,800 per month ($57,600 per year) in order to accommodate more data and more storage.

Tuning and auditing an environment like the above can give some customers up to 50% (or more) improvement in performance, and often see 20-25% reduction in space. Even with a conservative 25% boost in your performance and slight reduction in data and storage would save almost $15,000 in the first year alone. Over 4 years this could translate to an estimated $75,000 in total savings in AWS costs just based on smaller data and performance enhancements. And this is just a small setup.



At Percona, we have worked with many customers where cost management was the primary need instead of performance optimization. Through performance optimization, we can often provide a significant cost savings. Especially with modern cloud environments, costs savings often can be realized immediately

by reducing the number of servers or storage in use, or the cloud environment configuration. Even if you have already made hardware purchases for your current workload, optimization after the fact can achieve much higher performance from existing hardware and reduce future expenses.

Cost is often the driving factor for moving to an open source database. Many come to Percona for help moving from Oracle® or Microsoft SQL Server® to MySQL—not because MySQL performs better for their workload (it often doesn't), but because it offers acceptable performance at a dramatically lower cost. It also avoids vendor lock in—allowing for better pricing options for future needs.

There are many different types of costs businesses care about. Some of them are straightforward, such as hardware, licensing, or cloud services subscriptions. Other costs, however, are also very important—such as development and operations costs.

We've already discussed agility, and the time to implement new features. Since time is money, however, the larger the team required to build the same application, the higher the costs. Operational costs function the same as development costs. How much do you spend on the operations team to keep the database up and running 24x7? Generally, the larger and more complicated your environment is, the more expensive it is going to be to maintain.

Security and Compliance

Security and compliance can be considered shackles that don't help performance agility, development agility, or resource usage, but are increasingly important in the modern world. They are a great concern due to increasing government technology regulation and increased attention to security and privacy caused by several high profile security incidents.

Security and compliance requirements are heavily tied to specific industries or companies; typically little can be done but follow them.

Many security and compliance requirements, such as secure connections, data encryption, and audit logging, can be looked at as application features, and as such be optimized for performance appropriately. You should ask questions such as how do these features impact performance? Why is the performance affected? Is it reasonable? Is there anything you can do to reduce the overhead?

Do Not Be Afraid to Negotiate

We often encounter a development team that has been tasked to produce something that is difficult, expensive, and complicated to implement. Yet nobody ever asks the question of whether the request is really worth the cost.

Not questioning implementation requests is fraught with peril, as it can lead to overcomplicated and inefficient architectures. Always examine the performance implication of a requested feature and see whether an alternative solution would be "good enough," but easier to implement.

For example, when you enter "the" in Google, you get about 25 billion matches. Of course, it's not exactly 25 billion matches, because computing the exact number of matches is expensive and provides no value (and is kind of silly). From a user perspective, the difference between "about 25 billion matches" and "25,302,434,121 matches" is meaningless. Yet many smaller search engines insist on reporting the exact number of matching documents.

Sometimes you can redefine features so that you can make it "good enough" for the context, but much cheaper to implement. In other cases, you might opt for removing some features from the roadmap (or even existing features) if maintaining them with high performance is costly.

There Are Always Politics and Team Dynamics

As much as many of us would like to avoid politics, many organizations (especially larger ones) come with strong and persistent politics that span all areas—including choosing technology and making technology decisions. Whether you're working as consultant, or you're a permanent member of the team, it is important to know the political situation, understand what suggestions are feasible, which are taboo, how to present ideas, and how to make things happen. You have a limited amount of time and energy and have to pick your fights. Happily, most problems have more than one solution—there is always "more than one way to skin the cat." Often you can meet performance (or other) goals without ruffling too many feathers.

Remember to take team skills and experience into account. Solutions must meet the skills and strengths of the people that employ them.

For example, when working with a team with little experience using SQL and relational databases, we often end up choosing simpler queries and doing more work on the application. This isn't perfect, but it is best solution for the team.

On other hand, many Oracle users that now employ MySQL like using a lot of stored procedures. Stored procedures often don't perform as well in MySQL, and are not easy to debug. However, if not used excessively, they can work well.

In other cases though, some internal prejudices might be worth the fight. The infrastructure team of one customer we worked with was convinced by the DBA team that no one should need more than 16GB of memory, including databases. This philosophy ended up creating a huge IO workload that required a lot of servers. Since hardware selection was the infrastructure team's job, the DBAs were overstepping their boundaries. Convincing everybody to upgrade memory capacity to 64GB reduced the servers to a third of their previous number (including a reduction in the number of replicas) and increased response time at the same time.

If there is a lot to win, don't let politics stop you: pick a fight!

In the next section of this book, we'll begin discussing how to specifically isolate the cause of poor application performance, and how to work through various root causes in order to isolate MySQL as the issue.

How Percona Can Help

How Percona Can Help

Potential performance killers are easy to miss when you are busy with daily database administration. Once these problems are discovered and corrected, you will see noticeable improvements in data performance and resilience.

Percona Consulting can help you maximize the performance of your database deployment with our MySQL performance audit, tuning and optimization services. The first step is usually a Performance Audit. As part of a Performance Audit, we will methodically and analytically review your servers and provide a detailed report of their current health, as well as detail potential areas for improvement. Our analysis encompasses the full stack and provides you with detailed metrics and recommendations that go beyond the performance of your software to enable true performance optimization. Most audits lead to substantial performance gains.

If you are ready to proactively improve the performance of your system, we can help with approaches such as offloading workload-intensive operations to memcached. If your user base is growing rapidly and you need optimal performance at a large scale, we can help you evaluate solutions. If performance problems lie outside of MySQL or NoSQL, such as in your web server, we can usually diagnose and report on that as well.

Percona Support can provide developers and members of your operation team the 24x7x365 resources they need to both build high performance applications and fix potential performance issues. Percona Support is a highly responsive, effective, affordable option to ensure the continuous performance of your deployment.

Our user-friendly Support team is accessible 24x7 online or by phone to ensure that your databases are running optimally. We can help you increase your uptime, be more productive, reduce your support budget, and implement fixes for performance issues faster.

If you want to put your time and focus on your business, but still have peace of mind knowing that your data and database will be fast, available, resilient and secure, Percona Database Managed Services may be ideal for you. With Percona Managed Services, your application performance can be proactively managed by our team of experts so that problems are identified and resolved before they impact your business. When you work with Percona's Managed Service team you are able to leverage our deep operational knowledge of Percona Server, MySQL, MongoDB, MariaDB®, OpenStack Trove and Amazon® RDS to ensure your data (and your database) is performing at the highest levels.

Our experts are available to help, if you need additional manpower or expertise to improve and insure the performance of your system. To discuss your performance optimization needs, please call us at +1-888-316-9775 (USA), +44 (203) 6086727 (Europe), visit http://learn.percona.com/contact-me or have us contact you.

ABOUT PERCONA

Percona is the only company that delivers enterprise-class software, support, consulting, and managed services solutions for both MySQL and MongoDB across traditional and cloud-based platforms that maximize application performance while streamlining database efficiencies. Our global 24x7x365 consulting team has worked with over 3,000 clients worldwide, including the largest companies on the Internet, who use MySQL, Percona Server, Amazon RDS for MySQL, MariaDB and MongoDB.

Percona consultants have decades of experience solving complex database and data performance issues and design challenges. We consult on the full LAMP stack, from hardware to operating systems and right up through the database and web tiers. Because we are both broadly and deeply experienced, we can help build complete solutions. Our consultants work both remotely and on site. We can also provide full-time or part-time interim staff to cover employee absences or provide extra help on big projects.

Percona was founded in August 2006 by Peter Zaitsev and Vadim Tkachenko and now employs a global network of experts with a staff of over 125 people. Our customer list is large and diverse and we have one of the highest renewal rates in the business. Our expertise is visible in our widely read Percona Data Performance blog and our book High Performance MySQL.

Visit Percona at www.percona.com

PERCONA

Percona Corporate Headquarters

8081 Arco Corporate Drive, Suite 330
Raleigh, NC 27617, USA

usa +1 (888) 401-3401
eur +44 (203) 6086727

www.percona.com
info@percona.com