

# A Brief Introduction To Goal-Driven Performance Optimization

Baron Schwartz and Peter Zaitsev<sup>1</sup>



Goal-Driven Performance Optimization is the method we use at Percona to deliver results quickly for our clients. It is a simple, universal process that anyone can use. Benefits include time savings, a defined termination condition, avoidance of distractions, measurable results, and proof when goals are unachievable.

Goal-Driven Performance Optimization is not the solution for everything. It is for optimizing an existing system's performance under its current load. There are other methods for optimizing systems that don't exist, predicting performance, planning for capacity, reducing overall load, and many other useful things. It is also not overly formal or mathematical; in our opinion great precision and mathematical modeling are unnecessary. Systems with performance problems are usually rapidly changing, hard to measure, and do not conform to empirical models, so high precision is neither achievable nor desirable.

A method is a good thing to have. Learning and practicing a disciplined process is the best way to develop the skills and experience needed to know when to break the rules. If you're interested in the science of performance optimization, then we recommend *Optimizing Oracle Performance* by Cary Millsap for further reading.

## Background and Definitions

- **Performance and Response Time.** Performance = response time, commonly abbreviated by  $R$ . This is the time required to complete a desired task. A common mistake is to focus on metrics such as CPU utilization or memory usage. This is resource consumption, not performance.
- **Throughput.** Throughput is measured in tasks completed per unit of time. However, throughput is not the reciprocal of response time, because of queuing, parallel execution, and other factors. Throughput usually increases as load (utilization) increases, but the increase is non-linear, and adding more load to a system will eventually cause throughput to decrease.
- **Capacity.** Response time also increases non-linearly as the load increases, until it eventually reaches infinity. The system's capacity is the point where load cannot be increased without degrading response time below acceptable levels. Response time at the system's peak throughput is usually unacceptable, and thus systems cannot be loaded to their maximum throughput.

## High-Level Process

1. **Choose Key Tasks.** Choose several of your business's most important tasks, such as the "add to shopping cart" page on your website. Rank them by importance.
2. **Specify the Goal.** Decide upon an acceptable  $R$ , variation, and measurement window for each task. Tasks usually have different response time requirements. AJAX responses must be fast; search pages can be slower; nightly reports can be very slow. Express  $R$  in seconds, with an acceptable variation as a percentile. For example, the task must complete in 2 seconds or less, 95 percent of the time. The 95th and/or 99th percentile are both common. Do not use average or maximum, because these make the goal either impossible to reach, or possible to achieve without satisfying business requirements. A 95th percentile permits bad performance 5% of the time, nearly an hour every day, so you must measure in short intervals and require the system to meet the goal in every interval. In our experience, 5 minutes is a good interval. Document all of the above

in a spreadsheet. This is your goal, expressed as an SLA (Service Level Agreement). Here's an example:

An example of an SLA (Service Level Agreement)

Task	95th Pctile R	99th Pctile R	Interval
Home Page	0.3s	1.0s	300s
Product Search	1.0s	3.0s	300s
Product Details	0.5s	2.0s	300s
Add to Cart	0.5s	2.0s	300s
Log In	1.0s	3.0s	300s

3. **Measure Current Performance.** You cannot improve what you cannot measure. Measure *user-visible R* as specified in your SLA, and add the results to your spreadsheet. You should measure in production; test systems will not yield realistic results. Sometimes the user-visible response time is impossible to measure, so you can measure at the middle-tier or back-end instead. You might need to instrument your application so you can measure.
4. **Find SLA Violations.** Determine whether the application meets the business's response time SLA in every interval. Be sure you don't miss "performance holes"—particular executions of a task that are more expensive than others. For example, your system might perform very slowly for a search on "mp3 player," but very quickly for other phrases. It is important to find these holes, so there are no special cases that consistently violate your SLA. If your application meets the SLA and there are no holes, then you are done with your performance optimization project.
5. **Optimize.** Optimize each task that violates its SLA. We explain the process for this later.
6. **Measure the Results.** Repeat the measurements and compare the before-and-after results. Optimizations can cause side effects such as performance regressions or collateral damage. For example, creating an index to speed up a query can slow down another query; or a task could use more of a resource such as memory, competing with another task.
7. **Iterate.** Repeat the *Optimize* and *Measure* steps until each task meets its SLA, or until further optimization is more costly than the benefit to the business.
8. **Write a Report.** Briefly document the before-and-after in your spreadsheet, and compare to the SLA. Explain how you achieved the results. If no performance improvement for a task was possible, or it was not justified by the potential ROI, document that.

## Why Tasks Perform Slowly

Before we explain how to optimize a task that violates its SLA, here's some background. A task is slow for two reasons: it is doing a lot of work, or it is being slowed by something else. Computers are state machines, so optimizing response time is a matter of reducing the time spent executing or waiting to execute. You can reduce execution time only by reducing the number of CPU instructions to execute. Reducing wait time is more complex, because waits can be caused by queuing delay, coherency delay, or both. A process is queued when it makes a request to a resource that is busy and cannot service the request. Coherency delay is caused by synchronizing shared state. Sometimes coherency delay and queue wait appear to be execution time because of implementation details, such as using spinlocks to wait on a mutex.

The best approach is to begin at the user, and follow the flow of actions through the system, optimizing from the user towards the backend. Do not be tempted to blame a specific component immediately. If you rush

<sup>1</sup>Thanks to Cary Millsap of Method R Corporation.

to the backend, you might revert to trial and error, and miss the chance to measure and understand what is really happening. For example, the response time a web application sees for a database query might differ dramatically from the query's real execution time on the database, due to factors such as network latency.

## How to Optimize a Task

In the following process we show you how to optimize single executions of a task. In practice, most executions of a task that performs badly are slow for the same reason, and it's common to aggregate them together and optimize them as a group.

*Definition.* A sub-task is a component of the task at the high level. For example, adding an item to the shopping cart executes sub-tasks such as database queries and application logic. When optimizing tasks, we will drill into sub-tasks, sometimes recursively.

1. **Decide How To Observe.** To measure the task's performance, you must observe it when it performs poorly. Either find a way to measure poorly-performing executions, or measure all executions and exclude those that perform acceptably. To avoid losing data due to outliers, filter less aggressively than your SLA specifies. For example, if you are interested in the 95th percentile performance, capture the worst 10% of executions, not the worst 5%.
2. **Capture Diagnostic Data.** Capture detailed and *properly scoped* data about every important sub-task in the task's sequence diagram. Proper scoping means that you measure only the work the task itself performs, not the entire system's work; and only during the task execution, not the entire session. If this is not possible, you might need to instrument the application. Instrumentation should measure where the application spends its execution time, key diagnostic information such as total wall-clock time and CPU time, time spent on external requests such as database queries and web service calls, and the queries issued to those resources. It is also important to record performance and status metrics from the application stack, such as the operating system, network, and database. Store the diagnostic data for analysis. Plain log files are often fine, but it is nice to be able to load the data into something that supports ad-hoc querying, such as a database.
3. **Generate a Profile.** Generate a tabular view of the diagnostic data, aggregated by sub-task. Include sub-tasks, total time consumed, the total's percentage contribution to the overall *R*, the number of times the sub-task executed, and average *R* per execution. Show "missing time" in the profile; this is the difference between the task's total time, and the sum of all the time measured for sub-tasks. Sort by *R*, descending. Here's an example:

Profile for the "Product Search" Task

Rank	Sub-Task	<i>R</i>	<i>R</i> %	Calls	<i>R</i> /Call
1	DB::fetch_items	8.98	48.5%	1	8.98
2	App::Render_item	6.95	37.5%	50	0.14
3	Sphinx::search	1.43	7.7%	1	1.43
4	[Missing Time]	1.01	5.5%	-	-
5	App::Show_header	0.11	0.6%	1	0.11
6	App::Paginate	0.03	0.2%	12	0.00

This profile is for one execution of the Product Search task. As we said earlier, it is common to aggregate many tasks together, so the profile might show more than one search. Be careful with aggregation, because it can mask important information about individual executions—especially performance holes.

4. **Prioritize the Profile.** Estimate the effort required to improve each item in the profile, and consider the maximum possible benefit of doing so. For example, optimizing Sphinx—or even eliminating it completely—cannot improve the example task's performance by more than 7.7%. Choose sub-tasks for improvement based upon the potential benefit, ease of improvement, and importance to the business. Treat the Missing Time entry the same as the others; if it is significant enough, you need more instrumentation. If no item is worth improvement and Missing Time is insignificant, skip step 5.

5. **Optimize.** Apply the following optimizations to each sub-task that you decide to improve. You might need to optimize recursively. We suggest applying these techniques in the order listed.

- **Eliminate Waste.** Wasted work is that which you can remove without impacting the business requirements. In web applications, we often find that unimportant features such as pagination are causing performance problems. In databases, archiving and purging old or stale data reduces the amount of data that has to be examined to execute queries.
- **Cache.** If the sub-task's product is reusable, and the cost of maintaining and checking a cache is lower than the cost of executing the sub-task, then caching is a solution to many performance problems. There are many ways to cache, such as using *memcached* or pregenerating summary tables in the database.
- **Tune.** Make the sub-task execute more efficiently. In the case of SQL queries, to tune means to examine the query execution plan and make changes such as rewriting the SQL, creating better indexes, or making schema changes.

6. **Assess Capacity.** If optimization does not improve performance enough, and you are able to drill down and identify an external resource such as CPU or disk as the root of the problem, then that resource might not have the capacity to execute the task as quickly as required. There are two reasons for lack of capacity, and your system-level statistics can often help you determine which is to blame.

- **Competition for capacity.** Other tasks on the system can consume enough resources to impact the tasks you are measuring. You can detect this by observing that the resource is performing more work than your task requests—typically much more. You might not be able to observe this directly; for example, a SAN could be attached to many servers at once. Optimizing competing tasks can reduce the load on the resource, effectively increasing capacity. You can also reschedule work to an off-peak time when there is less competition, or move it to another server that is not as busy. Rescheduling doesn't decrease load, but it improves *R*.
- **Inadequate capacity.** Calculate whether the total available capacity is enough to support your SLA, based on what you know about the resource. For example, if the disk's random access time is 5 ms, and the task requires 1000 random file reads that cannot be optimized out by tuning, then it is not possible to complete the task in less than 5 seconds unless you can make the disk accesses faster (e.g. upgrading your disks) or avoid physical I/O (e.g. add more memory so the operating system can cache the data).

## Conclusion

At the end of this process, you have optimized your system's important tasks for the best possible user-visible response time. You did that by measuring where these tasks spend their time, and reducing or eliminating that time. At this point your system is optimal in its current form, with respect to the business's goals and the potential return on investment of your efforts.

However, as we mentioned at the beginning of this paper, the system still might not meet the SLA. In this case, you might need improvements that are not possible with the existing system. Redesigning the system, using different technologies, or other methods might be necessary. For example, you might need to use a dedicated full-text search solution instead of the search functionality included with your database, or you might need to use a different database.

If you would like help with optimization, please contact Percona at <http://www.percona.com/> or by calling the following numbers during business hours in Pacific Time. In the USA, call toll-free 1-888-316-9775; outside the USA please dial +1-208-473-2904.