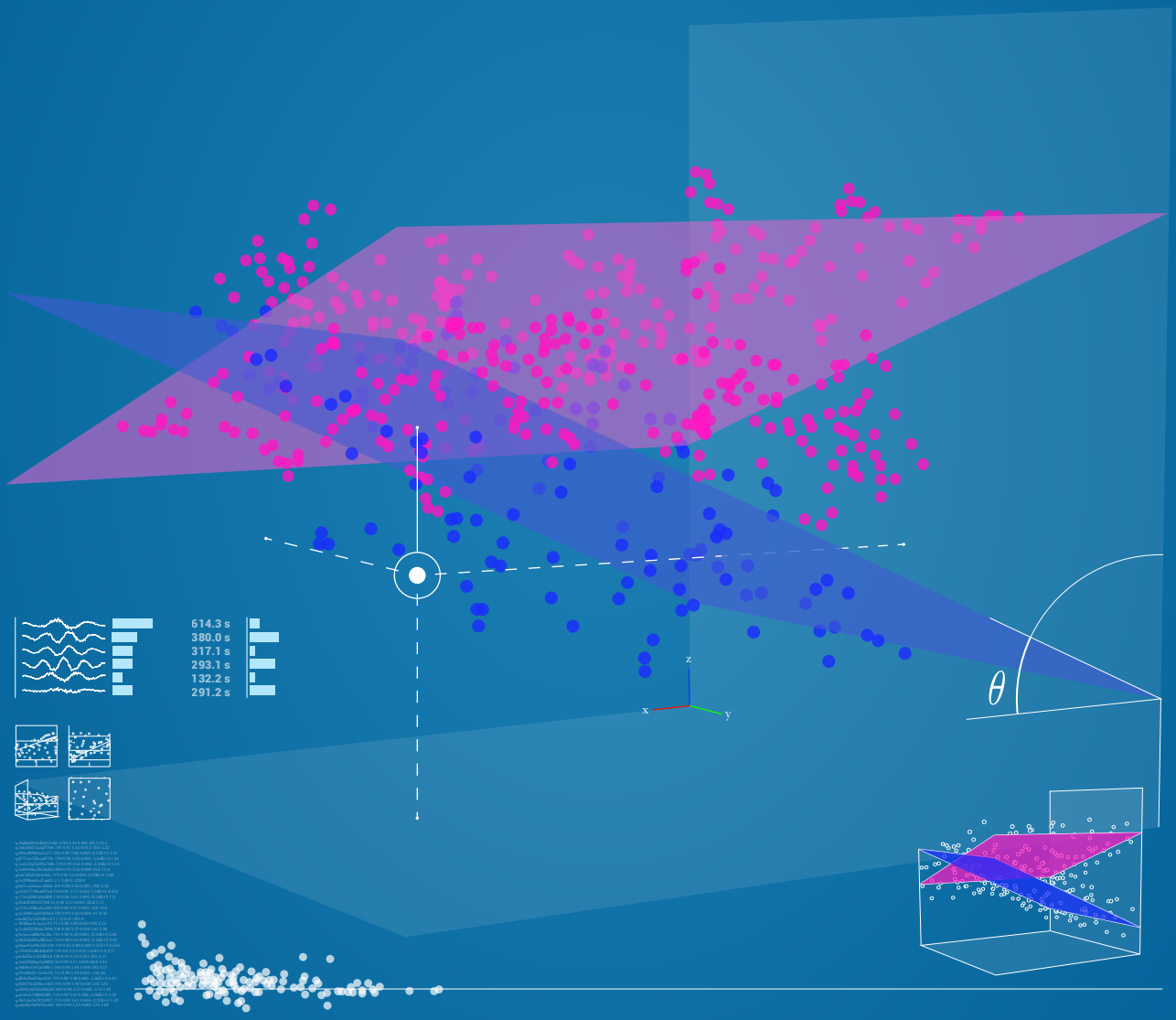
Estimating CPU Per Query With **Weighted Linear Regression**



November 19, 2016 • Revision 2

VividCortex

Meet the Author

Baron Schwartz

Baron is a performance and scalability expert who participates in various database, opensource, and distributed systems communities. He has helped build and scale many large, high-traffic services for Fortune 1000 clients. He has written several books, including O'Reilly's best-selling High Performance MySQL. Baron has a CS degree from the University of Virginia.

Table of Contents

- Optimizing Performance with Profiles ____ 3
- A Quick Regression Primer ____ 6
- Weighted Linear Regression ____ 8
- Solving Performance Problems with Regression ____ 11

Your database server is suddenly using a lot of CPU resources. Quick, what caused it? This is a familiar question for engineers of all persuasions. And it's often impossible to answer.

There are good reasons why it's hard to figure out what consumes resources like CPU, IO, and memory in a complex piece of software such as a database. The first problem is that most database server software doesn't offer any way to measure or inspect that type of performance data. The database server isn't *observable*. This problem arises in turn from the complexity of the database server software and the way it does its work, which actually precludes measuring resource consumption accurately!

But don't abandon hope yet, gentle reader! Statisticians have been getting answers to unmeasurable problems for generations, and we can do the same for database servers. In this ebook I'll explain how a specialization of ordinary linear regression can give useful answers. And *useful* is the watchword here: we don't need to know the answer for its own sake, we just need to solve the ultimate problem.

With the technique we've developed at VividCortex, you can easily identify which subsets of the workload on your database servers are consuming too many resources. Armed with that, you can fix them, and as a result you can get better performance from your servers, while potentially even decreasing your datacenter footprint or delaying additional hardware purchases.

After reading this ebook, you'll understand our special regression technique and how to use it to improve your database servers' performance and efficiency. This is a great example of why databases need specialized monitoring tools that measure their *workload*—the work they do (queries)—in high resolution and fine granularity. In a database, only query performance and resource consumption matter; most other things are auxiliary or vanity metrics.

Optimizing Performance with Profiles

Anyone who's done performance engineering on a database or large-scale application will appreciate how confusing it can be to troubleshoot performance issues. Successfully solving performance problems in an application that handles a complex workload is best approached through *profiling*, which categorizes similar types of work together and ranks them by some dimension.

Profiling lets you answer questions such as "which queries consume the most time?" and "which queries run most frequently?" as well as less obvious questions such as "if I make this query run twice as fast, how much can I improve the user's experience of the app's performance?"

Profiling is at the heart of VividCortex's approach to workload analytics. VividCortex lets you rank, filter, slice-and-dice, and drill down into your systems from the highest level (e.g. an entire datacenter summary) interactively, zooming into a single query execution on a single server with just a couple of mouse clicks. The primary tool for doing this is the profile.

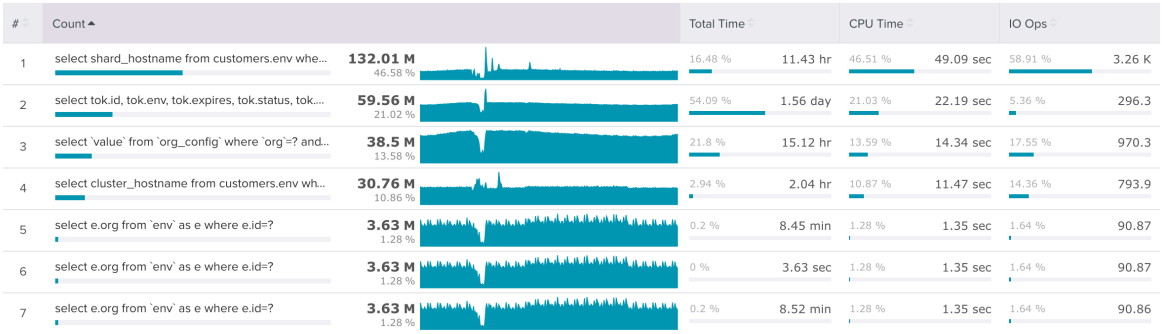


Figure 1: VividCortex's profiler feature, showing most frequent queries.

Figure 1 is an example profile, showing the most frequent queries in a database server. Each row in the table is a type of query. The left half of the profile shows statistics about query frequency: the total number, the percent of the total count of queries executed, and a graph of this metric over the time interval in question. The percent of the total is also shown visually, in the blue bar underlining each query's text.

Most typical profiles will show this information, but what typical profiles don't show is the additional columns with CPU Time and IO Ops. These columns, at the right-hand side of the profile, give additional insight into the resources each query consumes. You can see that the queries consume CPU roughly in proportion to how often they execute, but the IO demands for each query are quite different. The bulk of this server's IO load comes from the first query, and the second one uses almost no IO.

Now for the magical part: these quantities are *impossible to measure* within the database server. There's no system table or performance counter that reveals these, and indeed *there can never be*. This is because the database server (in this case, MySQL) performs a lot of work asynchronously in order to share resources and gain efficiency. Many queries' writes, for example, are aggregated and deferred to be done later. When that happens, a single IO operation might serve to write to disk the data that came from dozens or hundreds of different transactions. Because of this disconnect, accurate accounting of resources is impossible.

But with regression, you can still infer this cause-and-effect relationship and gain the insights you need: if you want to optimize your database server's use of IO, you should look at queries 1, 3, and 4, but not 2. (If you want to reduce CPU utilization and free up additional CPU resources, you should look at all of the top 4 queries.)

The rightmost two columns in the VividCortex profile screenshot shown in figure 1 are estimated from regression, not measured directly. The rest

of this ebook explains how we do it, because it's not vanilla linear regression—that doesn't work well enough.

A Quick Regression Primer

How can you determine CPU per query when you have per-query performance statistics, but only a global, aggregate CPU metric? This is the crux of the problem. You need to take an aggregate, server-wide metric of CPU or IO, and split it up into fractions which you blame on each type of query, effectively *disaggregating a global metric*.

The normal way to solve a problem like this is to use ordinary least-squares linear regression, which tries to figure out the best fit for a series of equations through a cluster of points in a multidimensional space. The general way to state this is as follows: given many samples of several independent (x) variables $x_1 ... x_n$ and a single dependent (y) variable, compute coefficients $c_1 ... c_n$ such that the equation $y = c_1 x_1 + ... + c_n x_n$ fits as closely as possible. "Fits" really means minimizing the sum of squared errors. For example, the result of a simple least-squares linear regression against the points in Figure 2 is the equation $y = 4.032x + 31.602$, which is plotted in the figure.

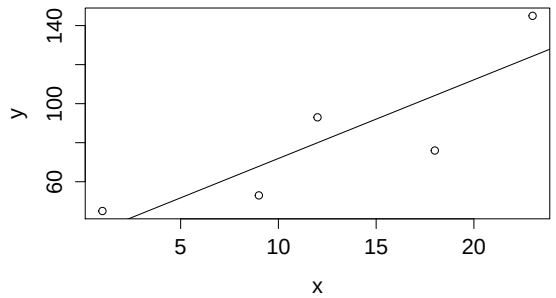


Figure 2: An example of simple least-squares linear regression

If you take this general approach and make it applicable to databases, you get hyperplanes through a multidimensional space. Each x-variable becomes the execution time (or count, or another metric) for each type of query, and y is the *total* observed CPU time, IO operations, or other metric of interest.

Linear regression is one of the dozens of techniques I researched and tried to solve this problem. It's not good enough, though. It's too complex, not efficient enough, and not accurate enough. This becomes obvious when you visualize some data from a sample dataset that comes from a real server. The problem is subtle, but essentially ordinary linear regression tries to figure out the relationship between each individual query, and the *total* CPU or other metric.

Let's plot that and see how it looks. The data in Figure 3 is execution time versus CPU time from a real database server. The first plot is a query that is one of the larger contributors to overall query time, and the second is a minority contributor.

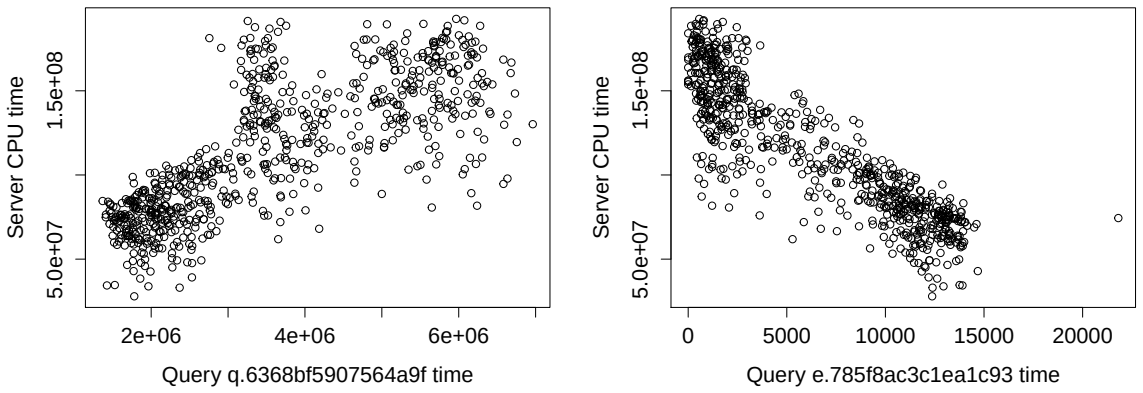


Figure 3: Query time versus CPU time

Ordinary linear regression is going to "draw a best-fit line" through each of those plots, and determine that the second one should have a negative slope. In other words, longer the second query executes, the less CPU it consumes. This is unphysical. Linear regression produces this type of impossible answer frequently in real systems, and it's why I developed a special type of regression to get better results.

Weighted Linear Regression

Fixing this and getting better results is a complicated topic that took months of research and development. In a nutshell, I examined dozens of different techniques to try to get good results—everything from machine learning to wavelet transforms to random forests to expensive proprietary software—and ended up coming full-circle back to regression, with some insights as to how to do a better job. I also had a benchmark, of sorts, that represented the best possible results from existing systems. The VividCortex algorithm outperforms that state-of-the-art solution, in both accuracy and runtime.

Some observations that led us to the solution:

1. The query metrics in the profile table all have the same dimensions and meaning: microseconds of time (or frequency). As a result, you can reasonably add, subtract, divide, and multiply them together.
2. The coefficients of the query cost equations are not likely to differ widely (by orders of magnitude), and are likely to be linearly related to metrics such as the server's CPU consumption. This is intuitively justified by observing that in many cases, queries within a database server spend much of their time executing CPU instructions. It seems reasonable to suggest that a query that runs twice as long as another one will be likely to execute about twice the CPU instructions, provided utilization is not so high that queue time becomes a significantly

nonlinear effect.[1]

3. In most cases a database server that experiences no demand from queries will eventually quiesce (perhaps after finishing some deferred work) and consume no CPU time. This is equivalent to saying that queries are directly or indirectly responsible for the server's *entire* CPU time consumption.

The key insight to getting better results is this: instead of performing regression between each query's metrics and the total CPU or IO metric for the server, it works better to go frame-by-frame in the time series of these metrics and assign a *weighted proportion* of the CPU or IO to each query present within that time frame, proportional to the magnitude of each query's metric.

More specifically, here's how it works. For each sample of y- and x-variables:

1. Assume that the sum of the x-variables is wholly responsible for the y-variable.
2. Divide the latter by the former to obtain a ratio, whose meaning and dimension is *quantity of y-variable per quantity of x-variable*.
3. Multiply each x-variable by the ratio to obtain an estimate of the y-variable that should be blamed on it for this sample. Call this estimate a z-variable. Note that each x-variable now has a corresponding z-variable.

This leads to a transformation of the problem: instead of performing a single regression of many x-variables against the sole y-variable, you can perform multiple independent simple (single-variable) linear regressions of each pair of (x, z) variables to determine the best-fit slope and intercept for each class of queries.

[1] This is a simplification that turns out to work well in many cases. In servers that are IO bound, weighted linear regression has proven to be a good means of understanding how much IO is caused by various classes of queries. For more on queueing theory and its nonlinear effects, see our related ebook.

How well does it work? Let's try it on the data used for Figure 3, now plotting each x-variable against its z-variable—its computed *contribution* to the y-variable instead of the entire y-variable. The result is Figure 4. The change in method makes it obvious that each query class's execution time has a positive relationship to the portion of CPU assigned to it.

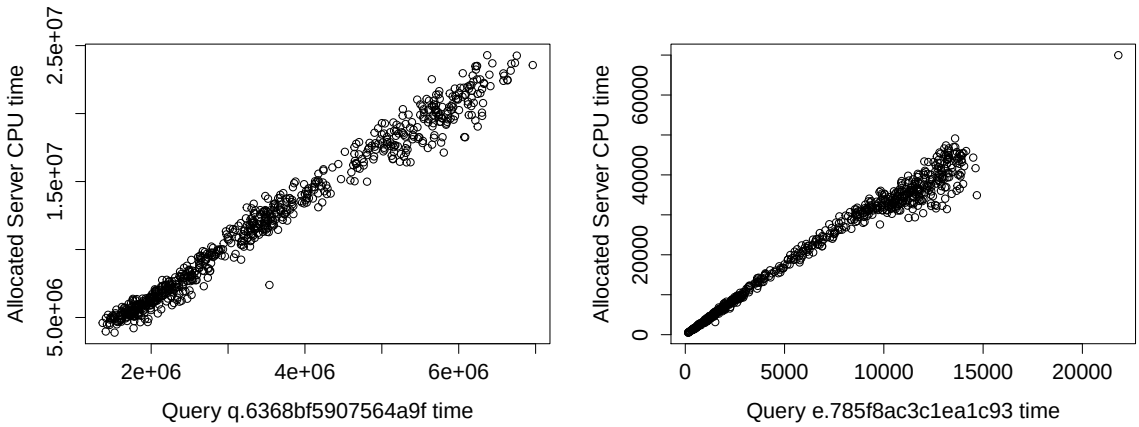


Figure 4: Query time versus weighted CPU time

Figure 4 is exactly the result I was looking for. It allows us to compute an estimate of each query's relationship to an external metric such as CPU or IO, even in the edge cases that ordinary linear regression doesn't handle well.

After testing and refining the algorithm widely on hundreds of real-world use cases, I added a few more nuances to handle special cases:

1. Non-positive slopes are disallowed. A query that runs longer should consume more CPU cycles, not fewer.
2. Positive intercepts are allowed. Queries may have a relatively fixed startup cost, such as query parsing and planning, and then an incremental cost that is proportional to its execution time.

3. Query classes that are not present in an interval (the x-variable is 0) are ignored during that interval, because a query that does not execute does not have a startup cost.

4. Query classes for which there are too few samples to perform an ordinary regression (fewer than three samples) are handled with straightforward Cartesian geometry.

I validated the algorithm's results extensively, and developed a vocabulary and toolkit for understanding how good a technique like this is. All regression is an estimate, and it's important for the people who are using it to know how to interpret it.

If you're interested in digging into the mind-numbing details of the algorithm and its results, I have provided sample source code, data sets, and a paper that I presented at the 2014 Computer Measurement Group conference. You can find all of these at the following Github repository: github.com/VividCortex/wlr.

Solving Performance Problems with Regression

I hope you're convinced by this point that solving performance problems with regression is useful. It lets you essentially measure the unmeasurable! Now, how do you put it into action?

Well, you could certainly use the sample source code I linked previously. But because this regression technique is built into the VividCortex user interface, it's trivial to use. Just use the "Choose Columns" control in the Profiler to add regressed columns to the profile, and use that to focus your optimization efforts.

Here are a couple of tips for the best results.

- A handy way to focus on the most important items in the VividCortex Profiler user interface is to click on the column heading. For example, profile queries by total time, add the CPU column, then click on the CPU column to sort the most expensive queries to the top of the table. (A similar trick applies with latency: rank by total time, then sort by Average Latency to sort the longest-running queries to the top from among the most time-consuming ones.)
- Regression works well when you're comparing things that have some relationship. For example, query execution time and CPU busy time are related—they are both measures of time. Similarly, frequency of execution is related to total execution time, because the more frequently a query executes, the more total time it accumulates. Sometimes regression gives good results for metrics such as rows accessed by queries, but not always—for example, there might not be much relationship between query time and rows accessed if queries spend most of their time waiting for I/O instead of reading rows. Similarly, regression doesn't always do a great job for yes/no variables such as whether a query created an intermediate temporary table on disk. Give it a try and see, but don't blindly trust the answers; always double check an EXPLAIN plan once you've isolated what looks like a query that uses temp tables on disk.
- Regression doesn't work as well for metrics that are aggregated across multiple servers. Typically, you'll see the best results when you filter down to a single host instance with the Host Selector. Different servers with different load and performance characteristics aren't "clean" enough to estimate well. Regression will always give an answer, but it might not be a trustworthy one!
- Because the algorithm assumes there's a linear relationship between x- and y-variables, it works best when servers aren't operating outside the limits of their roughly linear performance. That is, if they're operating at high utilization past the "elbow" in the queueing curve, the

results are less likely to be clean and trustworthy. For more on this, see VividCortex's ebook on queueing theory.

I hope you've found this ebook to be a helpful introduction to how regression can give you insights into things you couldn't normally measure directly. Now, go forth and optimize your servers!

About VividCortex

VividCortex is a SaaS database performance monitoring platform. The database is the heart of most applications, but it's also the part that's hardest to scale, manage, and optimize even as it's growing 50% year over year. VividCortex has developed a suite of unique technologies that significantly eases this pain for the entire engineering team. Unlike traditional monitoring, we measure and analyze the system's work and resource consumption. This results in better performance, at reduced cost and effort.

Related Resources From VividCortex

Best Practices for Architecting Highly Monitorable Applications

Is your app easy to monitor in production? This guide will help you architect for observability.

Database Performance Monitoring Buyer's Guide

This buyer's guide helps organize the database monitoring landscape so you can understand your needs clearly and select the best solution for your team.