PERCONA
Performance Consulting Experts

Forecasting MySQL Scalability with the Universal Scalability Law

A Percona White Paper

Baron Schwartz and Ewen Fortune

Abstract

Forecasting a system's scalability limitations can help answer questions such as "will my server handle ten times the existing load?" and "at what point will I need to upgrade my hardware?" Timely answers to these questions have more business value than exact predictions. Mathematical models can help reduce guesswork while avoiding the expense and time required for real load testing. Dr. Neil J. Gunther's *Universal Scalability Law* is such a model. It predicts a system's deviation from ideal scalability, based on simple measurements that are relatively easy to collect. In this paper we show how to model a MySQL database server's scalability, in terms of throughput, for two different servers and workloads.

The Universal Scalability Law is a mathematical definition of scalability. It models the system's capacity C at a given size N. This paper demonstrates how to use the Universal Scalability Law to predict a system's scalability. The process is to measure several samples of throughput and concurrency, transform the data, perform a least-squares regression against it, and reverse the transformation to find the parameters to the model.

The Universal Scalability Law has the following form:

$$C(N) = \frac{N}{1 + \sigma(N-1) + \kappa N(N-1)} \qquad (1)$$

The independent variable N is the number of users or processes active in the system.[1] Capacity is synonymous with throughput in this paper, so you can understand the model as a way to predict the database's queries per second at various levels of concurrency.

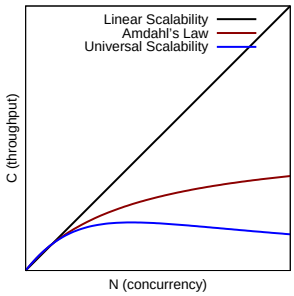
Equation 1 models the effect of the three C's:

1. The N in the numerator represents *concurrency*.
2. The σ parameter represents *contention*, or queueing: the system's performance degradation due to portions of the work being serial instead of parallel.
3. The κ parameter models *coherency* delay, or the cost of consistency: the work that the system must perform to synchronize shared data, such as mutex locking or waiting for a cache line to become valid. Coherency delay is caused by inter-process communication, and increases in proportion to the square of concurrency.

If $\kappa = 0$, then Equation 1 simplifies to the well-known Amdahl's Law:

$$C(N) = \frac{N}{1 + \sigma(N-1)} \qquad (2)$$

If both σ and κ are zero, then throughput increases linearly as N grows. The following graph illustrates linear scalability, Amdahl's Law, and the Universal Scalability Law. Notice that the Universal Scalability Law has a point of maximum throughput, beyond which performance actually degrades. This matches the behavior of real systems.



[1] When modeling hardware scalability, N represents the number of physical processors in the system, which is why this is a *universal* scalability model.

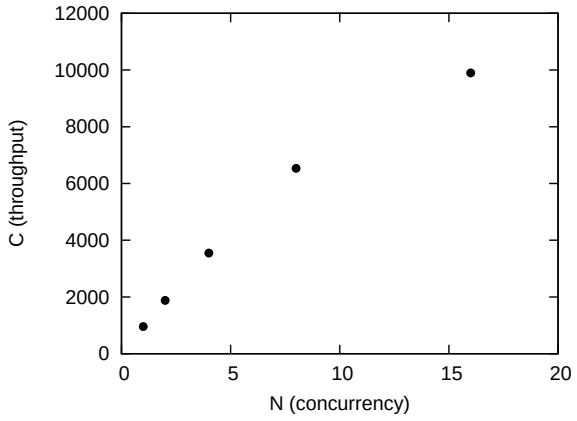
System 1: Cisco UCS Server

The rest of this paper illustrates how to apply the Universal Scalability Law to predict the scalability of two real systems. We will begin with measurements taken from a benchmark[2] that Vadim Tkachenko, Percona's resident benchmarking expert, ran against a Cisco UCS server with two processors, each of which has six cores. Each core can run two threads simultaneously, for a total capacity of 24 threads. The server has 384GB of memory and several high-speed solid-state storage devices.

This server's speed and capacity is representative of the next generation of commodity hardware, which will soon find its way into datacenters everywhere. We wanted to measure how well Percona's performance-enhanced version of MySQL scales on Cisco's leading-edge hardware. The software under test was Percona Server with XtraDB, version 5.1.47-11.2. Here are some of the results for the read-only workload:

Concurrency (N)	Throughput (C)
1	955.16
2	1878.91
4	3548.68
8	6531.08
16	9897.24

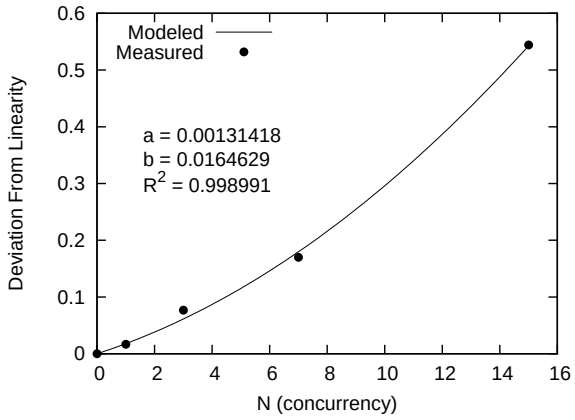
Our task is to determine the limit of the system's scalability—the point of maximum throughput. When plotted, these points appear as follows:



If the system had perfectly linear scalability, then $C(2)$ would be twice as large as $C(1)$, or about 1910 queries per second, but it is only 1878. There is clearly some non-linearity even at $N = 2$. The Universal Scalability Law lets us model this non-linearity by transforming the data, fitting a curve to it, and re-transforming the results. We will show all of the algebra later in this paper, but first we will show the steps in the process. The first step is to compute the non-linearity relative to $N = 1$. The following table adds two new columns[3] containing that computation:

N	C	$N-1$	$\frac{N}{C/C(1)} - 1$
1	955.16	0	0.0000
2	1878.91	1	0.0167
4	3548.68	3	0.0766
8	6531.08	7	0.1699
16	9897.24	15	0.5441

If we plot those columns, and fit a second-order polynomial of the form $y = ax^2 + bx + 0$ to them with least-squares regression, we obtain the following:

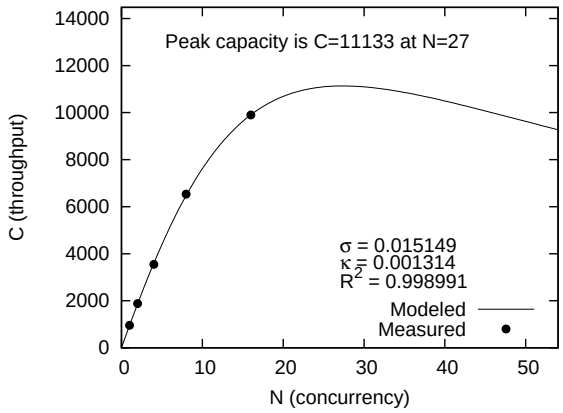


The R^2 value of more than 99% shows that the points fit the curve well. The coefficients of the polynomial are $a = 0.00131418$ and $b = 0.0164629$. We can use those coefficients to find the σ and κ parameters for the Universal Scalability Law. The relationship between the coefficients and the parameters, which again we derive later, is as follows:

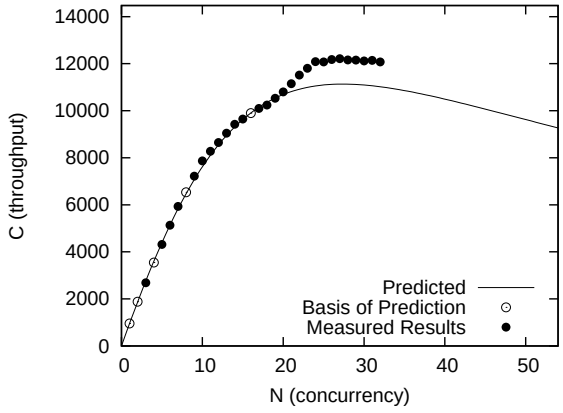
[2]See http://www.mysqlperformanceblog.com/2010/09/29/percona-server-scalability-on-multi-cores-server/ for more details.
[3]See Equations 6 and 7

$$\kappa = a \quad (3)$$
$$\sigma = b - a \quad (4)$$

The resulting parameters are $\sigma = 0.015149$ and $\kappa = 0.001314$, plus or minus some uncertainty whose derivation we omit for brevity. Plotting the Universal Scalability Law with those parameters, and the actual measurements, yields the following graph:



The model predicts that the system under test will reach its peak throughput of 11133 queries per second at a concurrency of 27. How good is this prediction? To find out, we can plot the full dataset. We based the prediction on measurements at powers of two, but Vadim's benchmarks actually included every concurrency level from 1 to 32. Including the rest of the data results in the following graph:



The prediction is not completely accurate, but the model matches the measurements fairly well. It is a good question how to explain the deviations beyond $N = 20$ or so. There might have been some errors in the benchmark procedure, the measurements, or even the benchmark software itself. We have not investigated this yet. Regardless, we feel confident in predicting that this system will not scale beyond 27 or so threads on this workload. In fact, the system's throughput peaks at 12211 queries per second at 27 concurrent threads, and subsequent benchmarks with many more threads proved that 27 threads was indeed the peak capacity.

System 2: Virtualized Dell Server

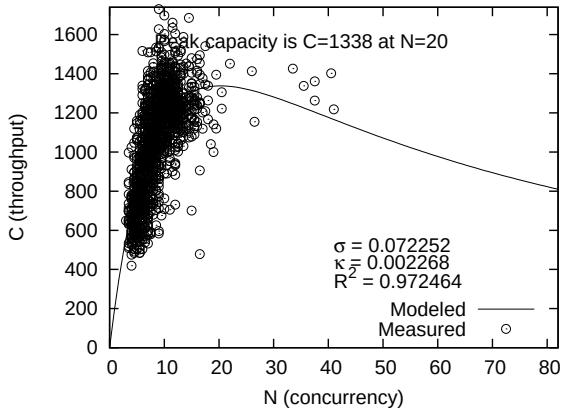
We obtained another dataset from a customer's MySQL server on a live production application, during the heaviest traffic of the year. We sampled counters from MySQL's `SHOW GLOBAL STATUS` command at ten-second intervals and recorded the following counters: `Questions`, `Threads_running`, and `Uptime`. These counters are the number of queries the server has received, the number of queries being executed at the instant the sample is taken, and the system's uptime in seconds. How close was the server to its peak capacity?

First, a few notes on the hardware and software used. The physical server was a Dell Poweredge 3950 with 16 CPUs, dedicated to a single VMWare ESX virtual machine. The database server was an unpatched MySQL 5.0.51a. All of the data was in InnoDB tables, and as a result of previous testing, Percona had set `innodb_thread_concurrency=8` to limit internal contention. The version of InnoDB included in unpatched MySQL 5.0.51a has well-known scalability limitations, so it was not a surprise that this artificial throttling was necessary to reduce contention. During the times of heaviest traffic, response time for certain queries began to increase noticeably, so our intuition was that the server was near its peak usable capacity.

We transformed our collected data as follows. For each sample, we subtracted `Uptime` from the previous sample's value to find the elapsed time, and subtracted `Questions` to find the queries per second

during that period. We averaged the measurement of `Threads_running` at the beginning and end of each sample query, and used that as the concurrency for the sample.

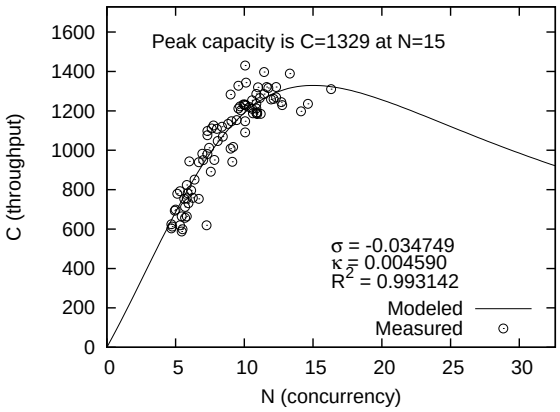
The full set of data we collected is comparatively large for applying the Universal Scalability Law. The usual practice is to capture a fairly limited set of measurements, perhaps between 4 and 50. We captured many thousands. Can we apply the Universal Scalability Law to this data? We began by simply trying to see what it looked like:



Both the plotted data and the model's prediction look unrealistic. The model predicts a peak capacity of 1348 queries per second, but we know the system can do more than that; we measured over 1700 QPS in some samples. And the model predicts this throughput at concurrency 24, which looks very wrong. The points at high concurrency, such as 40 or over, must be outliers, and they are skewing the results. We believe that it is impossible for this system to operate at such high concurrency. The throttling we applied to InnoDB ensures that real concurrency is limited to 8, and this server has only 16 CPUs.

So is this data clean enough to be usable, or are we trying to model garbage? We decided that we could not use this data as-is. The sampling period was too short, the workload was too variable, and the instantaneous measurement of `Threads_running` was likely to differ greatly from sample to sample. The system was also under heavy load and was experiencing many momentary spikes of high concurrency as miniature pile-ups constantly built up and cleared, a hallmark of earlier versions of InnoDB under these types of circumstances. As a result, although the measured throughput in each sample is reasonably reliable, the concurrency is very inaccurate.

However, it might be more useful to approach the dataset in the following manner: measure the throughput over longer intervals, and average the included measurements of `Threads_running` within the intervals. This will shift the problem from measuring nearly instantaneous bursts of throughput to a focus on sustained throughput over longer periods. We transformed the data again, this time over 150-second intervals, and obtained the following result:

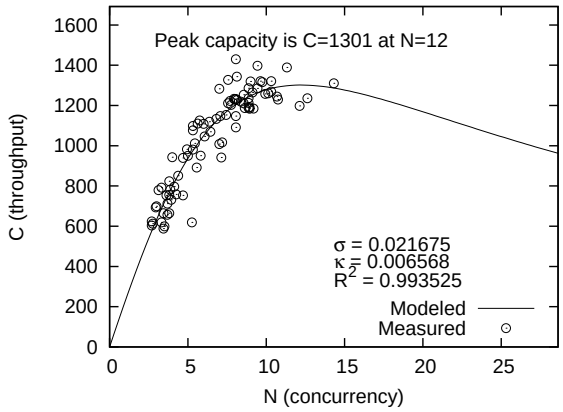


This looks much more reasonable, although it still looks wrong. The σ parameter is negative, and it looks like $C(1)$ is probably overestimated. But we can work with this, and the transformed dataset is much smaller, so it is not as hard to clean up the outliers.

We also need to make some corrections to the concurrency values, because the measurements are wrong. The first correction is that `Threads_running` includes the thread executing `SHOW GLOBAL STATUS`. This is a very different query from the application's workload. It must have some impact on the server—measurements always have some effect—but it is so small and short-lived that its true contribution to the server's concurrency is approximately zero. Therefore, we need to subtract 1 from the concurrency. In addition, three replicas are connected to the server, executing the

`BINLOG DUMP` command to subscribe to the data changes. Again, these threads are not really doing much work. The result is that the concurrency is overstated by a total of 4.

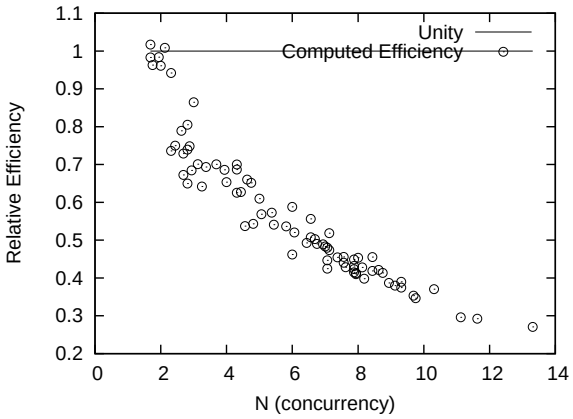
First things first—we fixed things that were clearly erroneous. After adjusting the concurrency downwards, the modeling immediately looked better. Visual inspection of the plot shows that the data "points towards the origin" more directly:



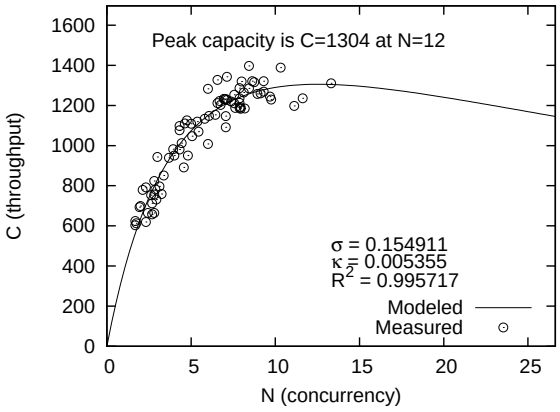
It would not be unreasonable to say that this is good enough, given the chaotic input data. The server's workload was mixed and variable, so great precision will never be possible. However, there are clearly outliers, and when you are faced with a dataset such as this, removing outliers is an important step in the process. You can take it too far and artificially mold your dataset into false results, but if your experience and judgment tells you that a specific data point is the result of something outside of the model, you should remove it (and document the removal).

Techniques such as plotting the residual errors can be helpful in identifying the outliers, as well as letting you verify visually that there is no particular pattern to the residual errors. Such a pattern might indicate that the data are not really appropriate for least-squares regression modeling against the polynomial. We investigated and removed the worst 7 points in the dataset, and improved the R^2 value for the regression against the polynomial from 97.4% to almost 99%. It might be even better to go back to the source data and remove the outliers from it, instead. However, we decided that was not required.

Another sanity check is to inspect the system's efficiency as concurrency increases. We plotted the scalability relative to $C(1)$ times the concurrency. For example, at a concurrency of 8, the system is producing only about 0.4 times as much throughput as $C(1) \times 8$:



The graph has some points whose relative efficiency is greater than one, which represents better-than-linear scalability. This impossible result is because we had no direct measurement for $C(1)$, so we interpolated downwards from the rest of the dataset. However, linear interpolation is almost certainly wrong; it assumes that the system scales linearly between $N = 1$ and the lowest concurrency for which we have measurements. We adjusted the estimate of $C(1)$ upwards by 2%. Here is the result, with outliers removed and the estimate of $C(1)$ adjusted:



Although our adjustments to this dataset took a while to explain, they were fairly small, and given

our knowledge of the system we measured and its workload, we had no trouble justifying them. To summarize,

- We averaged 15 samples together to derive each data point
- We adjusted concurrency to remove inflation introduced by the observer thread and connections from replicas
- We discarded 7 outlier data points caused by query pile-ups
- We adjusted our estimate of $C(1)$ upwards by 2% so that it did not show better-than-linear scalability for some data points

The resulting maximum predicted throughput of 1304 queries per second at a concurrency of 12 would have been difficult to predict, even for a very experienced person, but it is easy for us to believe that it is accurate. If this server experiences a load such that `Threads_running` approaches 10 or 12 (excluding replication slaves), then it will be delivering very poor service indeed to its users.

Systems cannot run at their maximum capacity. As they approach their limits, service times become unacceptably large and variable. So we would advise this client not to rely on a sustained throughput approaching 1300 queries per second with this application as currently configured, but at the same time we would not hesitate to say that no matter what, their server is simply incapable of more than this.

The result of 1300 queries per second is highly specific to this application, not indicative of what this server is capable of under other workloads. Although 1300 queries per second is not much, the most important queries in this application are complex many-table joins on large tables. They are slow, often running multiple seconds each.

Getting Better Results

The preceding section showed an example of successfully predicting a system's limitations based only on samples of `SHOW GLOBAL STATUS` from the MySQL database server. The results are not always this good. It works in this case because of the system's workload, which tends to create relatively high `Threads_running` values. In many systems, this is not the case. For example, we recently measured an Amazon EC2 server running Percona Server 5.1 that achieved over 20,000 queries per second for a typical Web workload, but our measurements of `Threads_running` were quite low, usually no more than 2 or 3. In such cases, the final data looks almost completely random when plotted, because sampling such low values for concurrency is highly inaccurate. In our experience, it is usually not possible to put such data into the model.

What tends to work better in such cases is to measure throughput and concurrency at the network layer, which is easy to do with acceptable accuracy. We have found very good results in practice, even better than those shown in our second example in this paper. However, explaining the process of gathering and transforming the data from the network is a topic for another white paper.

Another tactic that often works well is to match the throughput to metrics that are familiar to the business stakeholders, such as "number of people actively using the website." When possible, this can not only produce a better alignment with the scalability model, but the results are more meaningful to the business. It does require some care to ensure that the way the business measures users is valid, however!

An important note: before plotting the data as we have shown in this paper, it is best to plot time-series graphs of raw throughput and concurrency, and inspect the data visually. A dataset that includes widely disparate traffic, such as the normal Web traffic during the day and backup jobs at night, will certainly skew the results. A visual examination can make it obvious when to sample only a subset of the data.

Determining the Scalability Parameters

Now that we have shown how to predict capacity on real systems successfully, we will explain some

of the mathematical background, beginning with the parameters to the Universal Scalability Law, σ and κ. As you have seen, these parameters are not constants. They vary from system to system, and must be discovered empirically.

It is not difficult to compute the parameters. It is possible to transform Equation 1 so that the denominator takes the form of a second-degree polynomial, and to apply a matching transformation to the input data. This enables the least-squares regression we used to find the coefficients of the polynomial, a and b, which determine the σ and κ parameters for Equation 1. The following derivation shows the steps in the algebra:

$$\begin{aligned} C(N) &= \frac{N}{1+\sigma(N-1)+\kappa N(N-1)} \\ \frac{C(N)}{N} &= \frac{1}{1+\sigma(N-1)+\kappa N(N-1)} \\ \frac{N}{C(N)} &= 1+\sigma(N-1)+\kappa N(N-1) \\ \frac{N}{C(N)}-1 &= \sigma(N-1)+\kappa N(N-1) \end{aligned} \tag{5}$$

If we now define the following substitution variables,

$$x = N-1 \tag{6}$$

$$y = \frac{N}{C(N)}-1 \tag{7}$$

We can transform Equation 5 as follows:

$$\begin{aligned} y &= \sigma(N-1)+\kappa N(N-1) \\ &= \kappa N(N-1)+\sigma(N-1) \\ &= \kappa(N-1+1)(N-1)+\sigma(N-1) \\ &= \kappa(N-1)(N-1+1)+\sigma(N-1) \\ &= \kappa x(x+1)+\sigma x \\ &= \kappa x^2+\kappa x+\sigma x \\ &= \kappa x^2+(\kappa+\sigma)x \\ y &= \kappa x^2+(\sigma+\kappa)x \end{aligned} \tag{8}$$

Equation 8 is very close to the form of the desired polynomial $y = ax^2 + bx + 0$. All we need to do is state the relationship between the coefficients of the polynomial and the parameters of Equation 1:

$$a = \kappa \tag{9}$$

$$b = \sigma+\kappa \tag{10}$$

To recap, if you use Equations 6 and 7 to derive x and y values from the source data, then you can perform the least-squares regression against them to find the a and b coefficients for the terms in the polynomial. After this is complete, you can find the values for the scalability parameters through Equations 3 and 4, and substitute those into Equation 1.

The relationships expressed in Equations 9 and 10 result in the following, which are identical to Equations 3 and 4:

$$\begin{aligned} \kappa &= a \\ \sigma &= b-a \end{aligned}$$

And finally, the formula for finding the point of maximum throughput:

$$C_{max} = \left\lfloor \sqrt{\frac{1-\sigma}{\kappa}} \right\rfloor \tag{11}$$

All of the algebra in this section is based on Dr. Neil J. Gunther's book *Guerrilla Capacity Planning*; none of it is original, although we presented some of it in a more step-by-step fashion than his book does.

When Is This Technique Applicable?

This modeling technique is good for predicting the peak throughput for systems that have a relatively stable mixture of queries, or better yet, a single type of query. It might not work well when the workload on the system is changing in nature, or when the proportion of queries relative to the whole changes as

the load varies. It is best at predicting what happens when everything but the load is held constant.

In many cases, a system's scalability must be considered in light of many factors simultaneously, such as planned changes to functionality, new hardware purchases, data growth, or potentially increased query complexity. The Universal Scalability Law cannot forecast the effect of those changes directly. Human judgment and experience still matters!

Why Not Use Queueing Theory?

Queueing theory offers the Erlang C function, which models how long requests into a system will queue at a given utilization. The response time of any request is composed of two parts: the *service time*, which is the time actually needed to process the request; and the *queue time*, which is the time the request must wait before being serviced. Although there are very good uses for this model, it is usually too difficult to use for performance forecasting in practice. It requires precise measurement of the service time, which is notoriously difficult to measure in real systems. It also requires a specific distribution of arrival times, a constraint that is also difficult to measure and often not satisfied. In contrast, the Universal Scalability Law requires only a few data points that are usually easy to obtain, as we have seen.

In addition to the difficulty applying the Erlang C function, queueing theory is an incomplete model. Queueing alone cannot explain retrograde scalability; that can only be explained by inter-process communication, or coherency delay. Look again at Amdahl's Law, which models queueing: it has no maximum, which illustrates that it is not a complete model of real behavior. The missing element in Amdahl's Law is coherency delay.

Summary

The Universal Scalability Law is a model that can help predict a system's scalability. It has all of the necessary and sufficient parameters for predicting the effects of concurrency, contention, and coherency delay. It applies equally well to hardware and software scalability modeling.

To use the model, you must obtain some measurements of throughput and concurrency, preferably including $N = 1$. You then transform the data as shown in Equation 6 and Equation 7, plot the deviation from linearity (as compared to $C(1)$) in the result, perform a least-squares regression to fit the data to a parabola, and note the coefficients of the terms in the polynomial. You substitute the coefficients into Equation 3 and Equation 4 to determine the parameters for the Universal Scalability Law. Substituting these into Equation 1 enables you to predict the system's scalability at levels of concurrency beyond the data points you can measure.

We have omitted some details in this paper, in order to keep things clear and focused on our examples. In practice, you might need to perform additional steps to determine how reliable the input data is, and how much uncertainty there is in the results. The Universal Scalability Law requires some experience and judgment to use, but once you are familiar with the model, we think you will find that it is an invaluable tool for forecasting system scalability.

Acknowledgments

The Universal Scalability Law was discovered by Dr. Neil J. Gunther, an eminent teacher, author, and expert on system performance. Dr. Gunther's book *Guerrilla Capacity Planning* explains the Law and its theoretical foundations, with many more examples than we have shown in this paper. We highly recommend this book to anyone interested in performance forecasting or capacity planning.

Thanks to Cary Millsap of Method R Corporation, John Miller of The Rimm-Kaufman Group, and Percona's Vadim Tkachenko, Peter Zaitsev, and Espen Brækken for reviewing this paper. Their suggestions made it much better.

About Percona, Inc.

Percona provides commercial support, consulting, training, and engineering services for MySQL databases and the LAMP stack. Percona also distributes Percona Server with XtraDB, an enhanced version of the MySQL database server with greatly improved performance and scalability. If you would like help with your database servers, we invite you to contact us through our website at http://www.percona.com/, or to to call us. In the USA, you can reach us during business hours in Pacific (California) Time, toll-free at 1-888-316-9775. Outside the USA, please dial +1-208-473-2904. You can reach us during business hours in the UK at +44-208-133-0309.