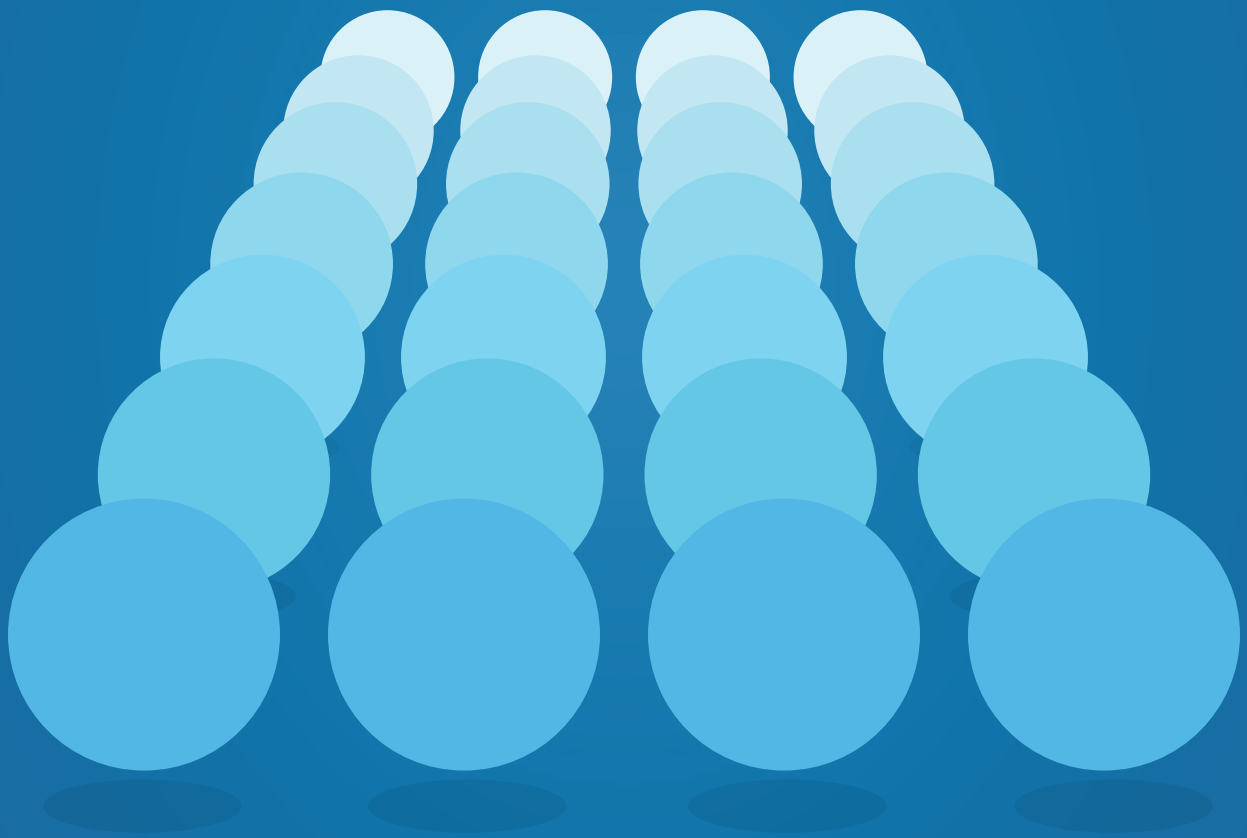
Everything You Need To Know About

Scalability



October 25, 2015

VividCortex

Meet the Author

Baron Schwartz

Baron is a database expert who is well-known for his contributions to the MySQL, PostgreSQL, and Oracle communities. An engineer by training, Baron has spent his career studying how teams build reliable, high performance systems, and has helped build and optimize database systems for some of the largest Internet properties. Baron has applied his systems thinking skills to both computer systems and teams of people, and has written several books, including O'Reilly's best-selling High Performance MySQL. Prior to founding VividCortex, Baron was an early employee at Percona, where he managed teams including consulting, support, training, and software engineering. Baron has a degree in Computer Science from the University of Virginia.

Table of Contents

Introduction

What is Scalability?

Scalability is ambiguous for many people—a vague term often tossed about in conference presentations, for example. Wikipedia's definition, borrowed from a 2000 paper by André B. Bondi, is "Scalability is the capability of a system, network, or process to handle a growing amount of work, or its potential to be enlarged in order to accommodate that growth." Although this isn't incorrect, it's still a bit informal, and this ebook needs a more formal definition.

Neil J. Gunther provides one such definition: scalability is a *function*. I read Dr. Gunther's books and heard him speak for quite a while before that sunk in for me. Scalability can be defined as a mathematical function, the relationship between an input (independent variable) and output (dependent variable).

What, exactly, that function *is* is a subject you can explore as deeply as you want. As you'll see later, I'll delve into it a little bit.

The important thing, though, is the correct understanding of the dependent and independent variables in the function. There are a few different ways to consider this. If you choose the variables incorrectly, you'll end up trying to analyze nonsense.

Bondi's definition provides a good clue: *work* is the driving factor of scalability. Useful ways to think about work include, to mention a few,

- units of work (requests)
- the rate of requests over time (arrival rate)

- the number of units of work in progress at a time (concurrency)
- the number of customers or users sending requests

Each of these can be sensible independent variables for the scalability function in different circumstances. For example, in benchmarks it's quite common to configure the number of threads the benchmark uses to send requests to a database. The benchmark usually sends requests as fast as possible, assuming zero think time, so the arrival rate is related to, but not strictly controlled by, the benchmark configuration (since it is determined by how quickly the database finishes each request).

One might say that load or concurrency is the input to the benchmark's scalability function, and the completion rate is the output.

In another scenario, you might vary the number of CPUs for the system under test (SUT) while holding constant the load per CPU, or if it's a clustered database, vary the cluster size and hold constant the load per node. In this case, the independent variable is the system resources.

In most cases I've analyzed, either concurrency or resources are usually sensible independent variables for a scalability function.[1] So for the purposes of this book, we'll consider scalability to be *a function of concurrency or capacity*. The dependent variable is usually the rate at which the system can process work, or *throughput*.

For those who are like me and need extra emphasis, I'll repeat that this is a mathematical function, with concurrency or capacity on the X axis, and throughput on the Y axis.

Now we have a sensible context in which to discuss the Holy Grail of Scalability: *linear scalability*. After all, linear means there's a linear relationship between the variables, which means there's a straight line when you plot them. If you hear people talk about linear scalability, which

[1] I have seen other scenarios, such as scaling by number of shards, but we don't need to dig into that in this book.

they often do in marketechture sorts of contexts, and it's not clear what the axes represent, you know there's a vague or poorly understood definition of scalability at play.

- Linear Scalability - must go thru origin, straight line - beware of points that look straight, don't show actual numbers - linear with 90% scaling factor - hardware and software scaling - different from eprformance

Nonlinear scaling

- serialization - crosstalk - common gotcha on benchmarks - nonlinear x-axis

The USL

- A formal definition - meaning of the coefficients - ahmdahl's law - hidden coeff

Fitting the USL to the real world

- collecting metrics - tcp data - mysql data - diskstats - cleaing data - removing outliers (faults) - regression - some examples

scalability capacity, and performance

performance is response time capacity is max work-getting-done with SLA of good perf (typically as percentile) by he way this is a problem with benchmarks - they push the system bad perf little's law r-time relationship

to usl https://www.desmos.com/calculator/vjlnruxxdi

$$y = \frac{1 + s(x-1) + kx(x-1)}{c}$$

is it valid? is r-time quadratic wrt concurrncy? if so then appd might be OK don't confuse this with r-time and utilization chart; concur -> infinity further reading: * https://groups.google.com/d/topic/guerrilla-capacity-planning/hei8zL2muuE/discussion * http://perfdynamics.blogspot.com/2015/07/hockey-elbow-and-other-response-time.html

capacity planning

* can we predict a system's capacity * queueing theory - hard because of service times * USL instead

Using the USL for capacity planning

- forecasting - show one of the datasets with only part of the data, does it predict the rest? - best and worst cases; repairman queueing - where is the lsos off linearity coming from - using it to see approx what % of capacity we're at now

the usl in real life

- how well does it work The USL is wrong: theoretical physicist, Richard Feynman: "In general we look for a new law by the following process: first we guess it. Don't laugh – that's really true. Then we compute the consequences of the guess to see what, if this law is right, what it would imply. Then we compare those computation results to nature, i.e.

experiment and experience. We compare it directly to observation to see if it works. “If it disagrees with experiment, it’s wrong. That simple statement is the key to science. It doesn’t make a difference how beautiful your guess is, it doesn’t make a difference how smart you are, who made the guess or what his name is – if it disagrees with experiment, it’s wrong. That’s all there is to it.” (Cornell lecture, 1964)

- * ceilings from hitting something’s max capacity like network tput * usually queueing causes retrograde to grow even faster than predicted - note that one could conjecture and analyze other shapes like the USL, e.g. https://www.desmos.com/calculator/nl53iwbngn

$$u(x) = \frac{bx}{1 + s(x-1) + kx(x-1)} \{x > 0\}$$

$$l(x) = \frac{bx}{1 + s(x-1) + k\ln(x)(x-1)} \{x > 0\}$$

$$r(x) = \frac{bx}{1 + s(x-1) + k\sqrt{x}(x-1)} \{x > 0\}$$

- Jayanta Choudhury has suggested changes to fit better.

superlinear scaling

- special cases: aggregate capacity not scaling proportionately to load, dataset, etc so each ”worker” has some advantage - special case: 1 and 2 nodes - effect of ”economies of scale,” that is a resource that is more efficient when shared than when used singly - https://queue.acm.org/detail.cfm?id=2789974

other PoV on scapability

- alternative definitions of scalability - new relic http://www.xaprb.com/blog/2013/01/07/a-close-look-at-new-relics-scalability-chart/

, appd, riak - cockcroft headroom plots * How to improve scalability * avoid crosstalk * avoid serialization * avoid queueing * Further reading * GCaP * Look at the Percona white paper

Example data * http://obartunov.livejournal.com/181981.html http://www.postgrespro.ru/blog/pgsql/2015/08/30/p8scaling scaling/scaling-postgrespro.png * VoltDB * mat keep https://blogs.oracle.com/MySQL/entry/comparing_innodb_to_myisam_performance https://www.percona.com/blog/2011/01/26/modeling-innodb-scalability-on-multi-core-servers/ * paypal example https://www.vividcortex.com/blog/2013/12/09/analysis-of-paypals-node-vs-java-benchmarks/ * example of robert haas http://rhaas.blogspot.com/2011/09/scalability-in-graphical-form-analyzed.html

About VividCortex

VividCortex is a SaaS database performance monitoring. 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 IT department. Unlike traditional monitoring, we measure and analyze the system's work and resource consumption. This leads directly to better performance for IT as a whole, at reduced cost and effort.

Related Resources From VividCortex

The Strategic IT Manager's Guide To Building A Scalable DBA Team

Case Study: SendGrid

VividCortex has been instrumental in finding issues. It's the go-to solution for seeing what's happening in production systems.