Tom Killalea, Amazon.com

Building Scalable Web Services

Build only what you really need.

In the early days of the Web we severely lacked tools and frameworks, and in retrospect it seems noteworthy that those early Web services scaled at all. Nowadays, while the tools have progressed, so too have expectations with respect to richness of interaction, performance, and scalability. In view of these raised expectations it is advisable to build only what you really need, relying on other people's work where possible. Above all, be cautious in choosing when, what, and how to optimize.

CAUTION: EARLY OPTIMIZATION

The first scalability-related meeting that I attended at Amazon had the title "Scaling for the Holidays." The date was June 3, 1998. Bob Vadrnais led the meeting, and for want of a meeting room the venue was his apartment. Bob could flawlessly execute diving saves that other engineers couldn't even visualize, and it was clear that surviving that holiday would depend more on heroic efforts than on an effortlessly scalable architecture. We discussed just two strategies at that meeting: the first was

bigger servers, and the second was tuning. It was a matter of scaling up to meet expected demand.

At that time the Amazon Web site was approaching its third year and had garnered traffic that for the time was impressive, but our focus was on customer experience and on growing as quickly as possible. The mantra was "get big fast." The emphasis on growth was appropriate, yet we understood that when the time came to focus on architecturally sound and sustainable scalability it would be all the more challenging.

Now, as then, the success rate of venture-backed technology startups is about 1 in 10. So if you're asked to build a *scalable* Web service, you might want to ask, "Are you sure?" Time and resources spent optimizing scalability might be better spent improving customer experience and driving traffic. It may be possible to settle for doing no harm or minimal harm with respect to future ability to scale as you build your site and grow traffic.

By the time we embarked on building infrastructure Web services—such as S3 (Simple Storage Service) and EC2 (Elastic Compute Cloud)—we had learned a lot about

Building Scalable Web Services

how large Web services scale, or sometimes don't scale, and we had a well-refined set of requirements.

EMBRACE: OTHER PEOPLE'S WORK

In the past 10 years the barriers to entry to building large, dynamic Web services have fallen dramatically. Three significant developments have contributed in different ways to this lowering of the barriers: the trend toward SOA (service-oriented architecture), the emergence of cloud computing infrastructure services, and the availability of Web application frameworks such as ASP.NET, Django, Rails, and Spring.

These developments greatly facilitate modularity and better separation of concerns. They also bring into focus a new separation of responsibilities. It's important to build only that which only you can build, relying on other people's work or services where possible. Time and resources spent building or customizing a Web application framework or building an infrastructure could be better spent improving the business logic. Everyone needs a framework and infrastructure, but they provide little incremental customer value, so time spent building them is time spent on undifferentiated heavy lifting.

CAUTION: OVEROPTIMIZATION

In Amazon's early days the events that pushed traffic patterns outside of the predictions of Gaussian models typically involved prominent coverage in national media. In the United States, the amplitude of the resultant traffic spikes was dampened by the spread of our U.S. customers across time zones—not everyone reads the paper or watches “Oprah” at the same time. When we expanded to single-time-zone markets with stronger social cohesion, such as Germany, prominent media exposure resulted in sharper peaks; however, even there the sharpest request rate transitions tended to be at the start and end of important football matches.

In a recent examination of types of decisions and the impact of different classes of randomness on outcomes, Nicholas Nassim Taleb revisited what he calls “black swans,” highly improbable and unpredictable events that have massive impact.⁴ He suggested that in areas of unpredictable (and frequently misunderstood) volatility such as financial and commodity markets, the drive for efficiency increases fragility. For example, commodity prices can double on a short burst in demand due to lack of slack. His recommendation: “Avoid optimization, learn to love redundancy.”

There is a strong focus on the optimization of capacity and maximization of server utilization among operators of large (and indeed small) Web services. In the current economic climate this focus will likely strengthen.

Unfortunately, Internet traffic patterns can sometimes appear to belong in what Taleb calls “Extremistan.” What was once termed the “Slashdot Effect,”—a traffic overload arising from many Internet users acting in unison—is now more common and of higher amplitude as a result of more continuous network access, increased closeness in social graphs, the adoption of metablogs, and the epidemic propagation of social network and smartphone applications. This is to say nothing of maliciously generated traffic floods. In addition, unanticipated events can impact the capacity supply side, with causes ranging from an inefficient software release to an external event such as the Northeast power blackout of August 2003. Redundancy is too often seen exclusively as a strategy to increase availability (by enabling routing around failed components). It should also be seen as a strategy to keep supply and demand in balance during an extreme event.

EMBRACE: THE CLOUD

Animoto is a Web application that uses artificial intelligence to generate professionally produced music videos. The company released it on Facebook last April, running on Amazon EC2. For days it ran on a mean of about 40 servers and a standard deviation in the single digits. As can sometimes happen with social network applications, Animoto suddenly became popular and within three days was using 3,500 servers. For this degree of scale-out, capacity must be elastic and free of friction; it's a bonus if you don't carry a cost or redundancy penalty for the slack when you're not using it.

CAUTION: TARGET-DRIVEN OPTIMIZATION

There are dangers in modeling expected traffic and then building a very precise scalability plan to meet that target. Good models are hard to build and can suffer from sim-

plifying or happy assumptions that discount variability. To quote Taleb, you can't "use models of uncertainty to produce certainties." The bigger danger is this: if your Web service is successful, you'll eventually see greater demand than the target model suggests—maybe not this Black Monday or Super Bowl Sunday, but perhaps soon after and at a less anticipated hour.

EMBRACE: RIPPING THE WINGS OFF

A scale test ideally runs on an environment that is indistinguishable from the production environment and runs until breaking point. One popular example of a breaking-point test is the "Boeing 777 Wing Load Test."¹ Beyond analyzing what broke first and why, we look for how the given application or service can make progress without the broken or missing pieces, and then rerun the test to determine the next breaking point, sometimes provoking comparisons with the Monty Python "Black Knight" scene.

One of the hardest problems in the operation of a large Web service is figuring out how to implement a high-fidelity test environment. There's no magic solution, but here again we've found that the problem becomes more tractable by using virtualized resources in our compute cloud. We can achieve a higher degree of fidelity (by running the same system images) and isolation, have greater agility in how we ramp tests up and down, capture results and problems for later analysis, and use resources more efficiently.

AREAS OF CONCERN

The most difficult challenge in building a scalable Web service is how to handle the trade-offs between durability, reliability, performance, and cost effectiveness in the presence of failures and extreme concurrent access patterns.

This challenge is particularly evident in the persistence tier. In this issue of *Queue*, Amazon.com's Werner Vogels discusses how embracing "eventual consistency" can help surmount the challenge of building reliable systems at a worldwide scale. In *Queue's* May/June 2008 issue Dan Pritchett of eBay looked at this persistence challenge from a different perspective.²

Craig Russell of Sun Microsystems discussed ORM (object-relational mapping) also in *Queue's* May/June 2008 issue.³ ORM is gaining widespread adoption and bringing scalability considerations. With ORM, data stored in the persistence tier is represented to the application as objects in the native object programming language, thus abstracting the persistence implementation details and facilitating better separation of concerns. An

inevitable side effect of using ORM is a lack of transparency with respect to the scalability and performance implications of how queries are ultimately constructed, how much data is retrieved, and at what cost.

In building the presentation tier it is necessary to facilitate scale-out in an automated, recovery-oriented, and unconstrained fashion. In his article, "High Performance Web Sites," in this issue, Google's Steve Souders focuses on front-end performance. He provides a detailed list of best practices, many of which, in addition to their latency impact, enable the front end to scale out more easily.

When it comes to the network you use to connect Web service components, there appear to be an unlimited number of ways to introduce complexity by having it perform "value-added" functions. Modern network devices come with many esoteric features. We try to keep the network simple, dumb, and cheap.

Finally, the path across the Internet, along which content is delivered, can affect the performance and ultimately the scalability of a Web service. In his article, "Improving Performance on the Internet," Tom Leighton of Akamai Technologies looks beyond the presentation tier to assess how transit capacity, latency, and reliability can affect performance, and he discusses content-delivery approaches to avoid the bottlenecks.

Practitioners responsible for building scalable Web services have challenges, expectations, and available technologies that have evolved considerably in just a few years. While there is a lot more to a modern Web service than was previously the case, much less of it needs to be built from the ground up. **Q**

REFERENCES

1. Boeing 777 Wing Load Test; <http://www.youtube.com/watch?v=pe9PVaFGl3o>.
2. Pritchett, D. 2008. BASE: an ACID alternative. *ACM Queue* 6(3): 48-55.
3. Russell, C. 2008. Bridging the object-relational divide. *ACM Queue* 6(3): 16-26.
4. Taleb, N. N. 2008. The fourth quadrant: a map of the limits of statistics; <http://www.edge.org/documents/archive/edge257.html#taleb>.

LOVE IT, HATE IT? LET US KNOW

feedback@acmqueue.com or www.acmqueue.com/forums

TOM KILLALEA has worked at Amazon.com since 1998 and is the vice president of technology with responsibility for infrastructure and distributed systems engineering.

© 2008 ACM 1542-7730/08/1000 \$5.00