



Wrapping it Up

Daniel A. Menascé • George Mason University

“In my end is my beginning. What we call the beginning is often the end. And to make an end is to make a beginning. The end is where we start from.” (T.S. Eliot, “Quartet No. 4: Little Gidding,” *Four Quartets*)

This is the 20th and last installment of Scaling the Web. I’ve really enjoyed this journey since starting the column in May/June 2002, and I thank my readers for their support and feedback throughout. My decision to stop now is based on my belief that we should constantly search for new beginnings through endings.

In all my articles, I aimed to provide a quantitative interpretation and analysis to distributed systems and Web-related issues.

This final column presents a review of issues I’ve discussed in these three-plus years. It provides a road map of previous articles aggregated into five major categories: middleware, site scalability and availability, distributed applications, autonomous computing, and workload characterization.

Middleware

Several of my past columns dealt with middleware-related topics, including communication models in middleware, Web services, grid computing, and peer-to-peer (P2P) systems. Two of the most important communication paradigms used in middleware solutions such as Web services are *asynchronous messaging* – in the form of message-oriented middleware (MOM) – and *remote procedure call* (RPC). MOM solutions tend to be more robust to failures than RPC, and they let service requesters continue to process while service providers work on their requests. However, programming MOM-based applications is more cumbersome because distribution isn’t as transparent to the programmer as with RPCs. My March/April 2005 column provided a quantitative framework for comparing MOM- and RPC-based solutions. This framework indicates that RPC is a better choice when the service provider’s response is fast, thus creating a negligible penalty for suspending exe-

cutation while waiting for a reply. MOM-based communication works better for long-lived transactions that require lengthy execution times from the service provider. Most real applications combine situations in which some service providers reply to the requester very fast while others take longer to reply. Thus, we must use both RPC and asynchronous messaging where appropriate.

In my November/December 2002 column, I started a discussion of quality-of-service (QoS) issues related to Web services and showed that we must analyze these issues from both Web service provider and user perspectives. I presented the notion of a *Web services flow graph* to help establish relationships between throughput for a Web service and other services it depends on in a composition scenario.

Subsequently, I addressed slow services’ impact on overall response time for transactions that use parallel Web services (January/February 2004). Using analytical models, I determined the slowdown of an application that uses several parallel Web services as a function of the slowest service’s slowdown and the number of services being used. I later discussed the QoS and cost aspects associated with composing Web services (November/December 2004), presenting a graphical notation we can use to represent various types of Web service composition scenarios – including any combination of probabilistic invocation, parallel invocation (fork), sequential activation, fastest predecessor-triggered activation, and synchronized activation (join). We can then use these Web service composition graphs to estimate a composite Web service’s average execution time and cost.

Many scientific and commercial applications that require numerous computational, network, and storage resources can benefit from grid com-

puting technologies and supporting middleware. Planning the capacity to guarantee QoS in such environments is a challenge because global service-level agreements depend on local SLAs. My July/August 2004 column provided a motivating example for grid computing in an enterprise environment and then discussed resource allocation's impact on SLAs. I also demonstrated how cost constraints can affect optimal resource allocation in a grid computing environment.

The traditional client-server model at the core of most middleware solutions tends to underuse the Internet's bandwidth and intensify the burden that dedicated servers face as their loads increase. P2P computing relies on individual machines' computing power and storage capacity to better use bandwidth and distribute this load in a self-organizing way. In P2P, nodes (or peers) act as both clients and servers, form application-level networks, and route messages (such as requests to locate resources). Peers maintain local directories with entries to the resources they manage, and can also cache other peers' directory entries. Many important P2P applications include distributed directory systems, new e-commerce models, and Web service discovery, which all require efficient resource-location mechanisms. In my March/April 2003 column, I presented a scalable solution to the problem, "given a resource name, find with a given probability the node or nodes that manage the resource." The solution lets us find a resource with a probability of greater than 90 percent while contacting less than 10 percent of the nodes.

Site Scalability and Availability

We can achieve scalability and availability for Web sites and Internet data centers by combining many technologies, including caching, clustering, and proper software design of the servers involved. Load testing is a common

way to evaluate a Web site's performance before it goes into production. To assess how a site supports its expected workload, load testing consists of running a specified set of scripts that emulate customer behavior at different load levels. Each emulated browser, or *virtual user*, must behave in a manner characteristic of actual users for a load test to be valid. My July/August 2002 column described how to conduct load testing, the QoS factors that it addresses, and how it satisfies business needs at several requirement levels.

High-volume Web sites often use clusters of servers to support their architectures. A load balancer in front of such clusters directs requests to the various servers in a way that equalizes, as much as possible, the load

placed on each. Two basic approaches exist to scaling Web clusters: adding more servers of the same type (scaling out) or upgrading the servers' capacity (scaling up). Which approach is better? The answer depends on the application's response time, reliability requirements, and cost constraints. In my September/October 2002 column, I used simple queuing-theory models to examine the average response time, capacity, cost, and reliability trade-offs involved in designing Web server clusters.

Several protocols and mechanisms are aimed at enforcing various security dimensions in applications ranging from email to e-commerce transactions. Most use encryption as a basic component, but adding such mechanisms and procedures to applications

and systems isn't cheap. Security is intrusive, increases the total cost of owning a computer, and can cause performance degradation due to encryption, which is compute-intensive. My May/June 2003 column described some security trade-offs regarding performance and scalability. My analysis was quantitative and illustrated how using a specific security protocol or set of cryptographic algorithms could affect a computer system's performance.

When a Web site becomes overloaded, customers can be frustrated by long response times or rejected requests. This situation can lead to undesirable revenue loss and tarnished reputations for organizations that rely on their Web sites to support mission-critical applications. In my

Security is intrusive, increases the total cost of owning a computer, and can cause performance degradation due to encryption, which is compute-intensive.

July/August 2003 column, I discussed how organizations can use Web site caching and content-delivery networks to improve their sites' performance and scalability. I also provided some simple quantitative expressions to help designers understand some important trade-offs.

A Web server's software architecture can affect performance significantly. Two dimensions generally characterize the architecture:

- The *processing model* describes the type of process or threading model used to support the Web server operation.
- The *pool-size behavior* specifies how the pool size of processes or threads varies with time and workload intensity.

My November/December 2003 column classified Web-server architectures, offered a quantitative analysis of some possible architectural options, and discussed the importance of software contention (waiting for available threads, for example) on overall response time.

Internet data center (IDC) users pay for the services they obtain, so they want them delivered according to established SLAs – for example, average response time shouldn't exceed two seconds for a given application, the IDC must be able to process an average of 8,000 requests per second, and a given application should be available at least 99.999 percent of the time. An IDC must provision enough capacity and redundant resources to

SLAs established with each component. My September/October 2004 column examined the problem of determining which response time SLAs should be negotiated at the component level to satisfy application-level SLAs with the least possible cost. I presented an efficient, easy-to-compute heuristic to this optimization problem.

We can use grid computing and virtualization to distribute computing applications on the Internet, letting interconnected servers host a set of applications to balance the processing load and improve performance. An important question is which applications to assign to which servers so we can meet the applications' QoS goals, given the servers' bandwidth and

a two-phase *generation cycle* consisting of data-product generation and a quiet period, when new data accumulates. Some of the scalability questions related to these architectures include:

- What is the maximum data-product retrieval rate from the DPGCs?
- What is the average data-product retrieval time for a given retrieval rate?
- What is the impact of the quiet period's duration on data-product retrieval time?

I answered these questions using simple analytic models.

Autonomic Systems

Modern computer systems, such as e-commerce sites, have complex architectures comprising multiple tiers of servers, each of which can have many components. The workload tends to be bursty in nature, especially at fine time scales. Networked systems often have several configuration parameters, whose settings can significantly affect a system's performance. Thus, we can't always react quickly enough to the workload's fast variations to determine which configuration parameters could optimize the QoS for a given workload's intensity. My January/February 2003 column covered how to design a QoS controller that continuously monitors an e-commerce site's workload and determines the configuration that best meets the site's QoS goals. At regular intervals, the QoS controller executes an algorithm that takes into account the observed workload, the desired QoS levels, and any performance data from the site's various resources (such as CPU and disks) to determine the configuration parameters' best values. I based this approach on combinatorial search techniques and queuing network models.

Along the same lines of using queuing-network models to dynamically control QoS, my March/April 2004 column discussed how to make software

Scientific applications pose significant challenges to scalability.

ensure that it can meet its performance and availability SLAs. Failing to do so can incur revenue loss or, in some cases, penalties. For example, as the number of machines in the IDC increases, the cost of operating the data center increases, as does the IDC's total throughput and availability. Additionally, as the IDC's repair staff size and skill level increase, the cost of operating the IDC increases, but it also ensures that failed machines return to the available pool faster, which improves the IDC's throughput and availability. In my May/June 2004 column, I used analytical models to analyze such trade-offs and explain the relationship between throughput and availability, which we measure as the probability that a given number of machines are in operation at the IDC.

Distributed Applications

Modern distributed applications comprise a set of components, running on different networked machines. Application-level SLAs depend on the

capacity constraints. In my January/February 2005 column, I precisely defined the problem and discussed a solution through a motivating numerical example.

Scientific applications, from space exploration to environmental science to high-energy physics, pose significant challenges to scalability because they generate huge volumes of data and require significant processing capacity and network bandwidth. In my May/June 2005 column, I considered an architecture – one typical of many scientific data-collection processes – in which satellites collect data and send them to various ground stations. These ground stations calibrate and validate the raw data before sending them to data product generation centers (DPGCs), which generate and transfer high-level data products over the Internet to end users that request them. Because DPGCs are constantly receiving new data, they must constantly generate new data products. These data centers typically go through

components QoS-aware in a way that lets them negotiate QoS requests and perform admission control to guarantee previous QoS commitments.

Workload Characterization

One of the first steps toward analyzing an e-commerce site's scalability is being able to understand the nature of the workload submitted to it. As I explained in my September/October 2003 column, we can characterize workload at four layers: business, session, function, and protocol.

At the business layer, we can examine a business's overall characteristics and how they affect the way users interact with the site. For an auction site, for example, we would be interested in how many bids the winner placed, when the winner placed his or her first bid, the closing price's evolution over the auction's life, and the percentage of auctions that have winners.

The session layer deals with characteristics such as duration, navigation patterns within a session, and the buy-to-visit ratio (the probability that a session will result in a sale).

We can use customer-behavior model graphs (CBMGs) to capture how customers invoke the various functions an e-business site offers. The function layer specifies what functions are offered and how often they are invoked. An auction site, for example, provides browse, search, register, login, view bid, bid, and sell functions.

The protocol layer describes the workload's features in terms of HTTP requests.

A typical online retailer's workload is well captured by the Transaction Processing Performance Council's TPC Benchmark W (TPC-W) for e-commerce. My May/June 2002 column described its main features, advantages, and limitations.

The future holds many interesting challenges for scalability and QoS at

the application and system levels. A special issue on application-level QoS for distributed applications, which I will be guest-editing with Murray Woodside of Carleton University, is scheduled for *IC*'s May/June 2006 issue. It will be an interesting forum in which to explore the engineering of QoS into distributed applications. Another area that's gaining significant attention is autonomic computing, a topic on which I'll be guest-editing *IC*'s January 2007 issue with Jeff Kephart of IBM Research. I hope to receive submissions from many of you for these issues. □

Acknowledgments

Grant number NMA501-03-1-2022 from the US

National Geospatial-Intelligence Agency (NGA) partially supports this work. I thank Munindar Singh, former editor-in-chief of *IEEE Internet Computing*, for inviting me to write this column and Bob Filman, current EIC, for his continued support. The work of *IC*'s editorial staff also deserves special recognition.

Daniel A. Menascé is a professor of computer science, codirector of the E-Center for E-Business, and director of the MS in E-Commerce program at George Mason University. He received a PhD in computer science from UCLA. Menascé is a fellow of the ACM and a recipient of the 2001 A.A. Michelson Award from the Computer Measurement Group. Contact him at menasce@cs.gmu.edu.



How to Reach IC

Writers

For detailed information on submitting articles, visit www.computer.org/internet/author.htm.

Letters to the Editors

Send letters to lead editor Rebecca L. Deuel at rdeuel@computer.org. Please reference the article and issue to which you are responding.

On the Web

Access www.computer.org/internet/ or *IEEE Distributed Systems Online* at <http://dsonline.computer.org>.

Subscribe

Visit www.computer.org/subscribe/.

Changes of Address

Send change-of-address requests for magazine subscriptions to address.change@ieee.org. Be sure to specify *IEEE Internet Computing*.

Article Reprints

For pricing or to place an order, contact internet@computer.org.

Missing or Damaged Copies

If you're missing an issue or received a damaged copy, please contact membership@computer.org.

Reuse Permission

For permission to reprint an article, contact William Hagen, IEEE Copyrights and Trademarks Manager, at copyrights@ieee.org.

www.computer.org/internet/