Scaling Web Sites Through Caching

Daniel A. Menascé • *George Mason University* • *menasce@cs.gmu.edu*

A large jump in a Web site's traffic may indicate success, but this increased interest could turn into a nightmare if the site is not prepared to handle the bigger load. Site resources (processors, storage boxes, load balancers, LANs, and the like) could see their utilization rise to levels that generate increasingly long response times. Eventually, one of these resources — the so-called bottleneck resource — will reach 100 percent utilization, pushing the site's throughput to its maximum point.

When a Web site becomes overloaded, customers grow frustrated with long waits and rejected requests. This situation can lead to an undesirable loss of site-generated revenue and may even tarnish the reputation of organizations relying on Web sites to support mission-critical applications.

In this article, I discuss how caching technologies can improve Web site performance and scalability. I'll provide some simple quantitative expressions to let designers understand the most important trade-offs.

What Is Scalability?

A Web site is said to be scalable if it provides adequate service even when the workload increases above expected levels.[1] We often measure service levels in terms of response time, throughput, availability, and business-oriented metrics such as revenue throughput. A scalable site should be able to handle significant increases in workload intensity by adding more resources without having to change its basic architecture. Many Web sites, for example, use clusters of Web servers that allow the site to scale by adding more servers (scaling horizontally) or upgrading existing servers (scaling vertically). In a previous column, I analyzed performance and availability trade-offs for Web clusters.[2]

The type of elements that make up a Web site, these elements' performance characteristics, and the way they're arranged determine a site's scalability. I use a typical transactional Web site (one that supports e-commerce applications) to illustrate the scalability issues discussed in this article.

Web Site Architecture

Figure 1 shows a typical transactional Web site's architecture, which is organized into multiple server tiers: Web, application, and database. Load balancers and cache servers deployed at the front end provide a more equitable distribution of requests to the Web servers and faster access to the most popular Web objects.

Incoming requests are initially directed to a cache server to determine if the Web object associated with the request is already in the cache. In such a case — a cache *hit* — the object is returned directly to the client. If the object is not in the cache — a cache *miss* — the request is directed to the load balancer, which determines the most appropriate Web server for the request.

Web servers directly serve static objects, such as static HTML pages, images, text, video, and sound files. However, in transactional Web sites, many requests require function execution to generate an HTML page. Consider, for example, a user submitting a search request to an online bookstore. The Web server passes the request to an application server, which executes an application that sends the request to the database server and then generates an HTML page with the search results. This page is returned to the Web server, which returns it to the client.

Application servers implement the site's business logic and are responsible for executing functions such as search, shopping cart operations, order processing, and payment processing. Executing most of these functions requires using the persistent data stored in a back-end database server.

 Published by the IEEE Computer Society 1089-7801/03/$17.00

Web Site Caching

Web resource caching is a powerful concept for reducing latency and saving bandwidth. We find caches at Web browsers, organization proxy server caches, Internet service providers, content delivery networks (CDNs), and Web servers.[3] A server-side cache's main purpose is to reduce the time needed to process a request. Server-side caches store recently requested documents that will likely be referenced again in the near future. Other layers of the site architecture do not need to process these cached documents.

We get reasonably large object-cache hit ratios by caching a relatively small number of objects: the frequency of reference f to Web documents is inversely proportional to the rank r, which we measure in terms of the document's popularity.[4] The most popular document has $r = 1$, the second most popular has $r = 2$, and so on. This relationship, called Zipf's law, states that $f = k/r$, where k is a constant. So, the number of accesses to the most popular document is equal to n times the number of accesses to the nth most popular document.

Let's examine how this property can help us determine which objects to cache in the server-side cache's server. Assume that D documents are stored at the site and that they are numbered from 1 to D in increasing order of popularity. The frequency of access f_i to document i is equal to k/i, according to Zipf's law. We get k by observing

$$\sum_{i=1}^{D} f_i = 1,$$

thus, $k = 1/H_D$, where the notation H_n stands for the harmonic number defined as

$$\sum_{i=1}^{n} 1/i.$$

If A accesses to the Web site occur over a certain time period, the number of accesses to the first m documents is

$$S_m = \sum_{i=1}^{m} A \times f_i$$

$$= A\sum_{i=1}^{m} 1/\left(H_D \times i\right) = \frac{A \times H_m}{H_D}.$$

If the m most popular documents are kept in a cache, the cache-hit ratio is

$$R_m = \frac{S_m}{A} = \frac{H_m}{H_D}.$$

Because the number of documents D is typically very large, we can use the following approximation[5] for H_D:

$$H_D \approx \ln D + 0.5772156649,$$

which means we can compute the cache-hit ratio as

$$R_m = \frac{H_m}{\ln D + 0.5772156649}.$$

Figure 2 (next page) shows how the cache-hit ratio varies as a function of the m/D ratio of documents cached for a collection of 100,000 documents. The picture shows that a small number of cached objects yields a surprisingly large cache-hit ratio. If 2,000 documents (2 percent) are cached, for example, we get a cache-hit ratio of 67 percent.

This result is an interesting motivation for building cache-initiated refresh and preload algorithms. The cache must keep track of the popularity of the objects referenced and periodically refresh itself to make sure it contains the m most popular objects. Clearly, we would see a deviation from this optimal situation in practice because some Web objects expire and object popularity shifts with time. We must adjust the cache-refresh period to keep the cache as close to optimal as possible. In this context, we should view Figure 2's curve as an upper bound on the object-cache hit ratio.

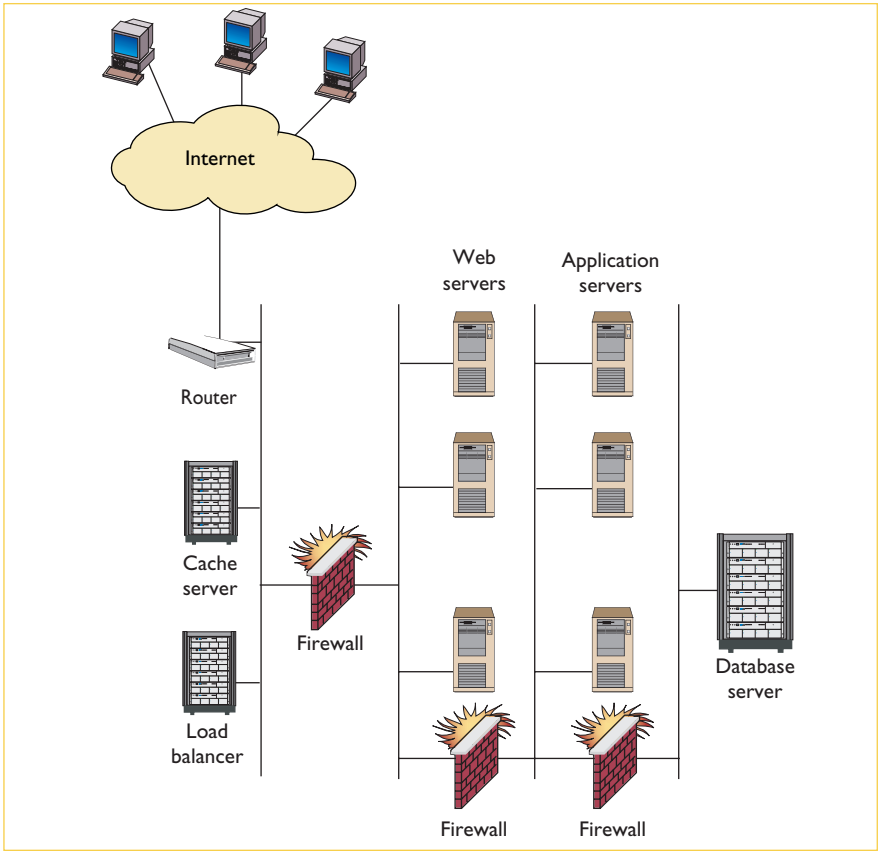

Figure 1. Typical architecture of a transactional Web site. The load balancer routes any requests the cache server can't resolve to one of the Web servers; application servers are involved in processing business rules, and the database server stores persistent data.

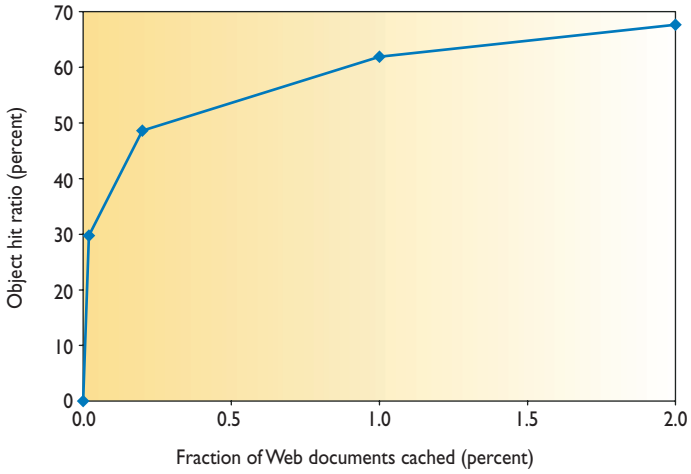

Figure 2. Object-hit ratio versus the fraction of objects cached. Two percent of the objects in the cache yield a 67 percent cache-hit ratio.

Table 1. File type and size breakdown.

File type	Percent	Average size (Kbytes)
Static HTML	65	18
Images	15	15
PDF	4	324
Video clips	3	12,000
Other downloads	1	1,336
Dynamic content	12	0

This analysis assumes that access frequencies follow an exact Zipf's distribution. Many empirical studies show that the access pattern to Web objects closely follows Zipf's law.[4,6] These studies show that the relationship between access frequency and document rank is of the form $f = k/r^{\alpha}$, where α is close but not exactly equal to 1. My colleagues and I showed that the access frequency to query terms in search requests on e-commerce sites also follows Zipf's law.[7] Caching the results of the most popular searches would be extremely cost-effective.

We also must consider other aspects when determining which objects to cache. One of them is the time involved in generating an object. Larger files, for example, require more I/O and processing, which generate and use more bandwidth. Thus, we might want to minimize the byte-hit ratio instead of the object-hit ratio. This is important because Web objects sizes have heavy-tailed distributions.

Content Delivery Networks

CDNs manage thousands of reverse-caching proxies around a network's edge; the CDN's caches act as surrogates for content provider servers. Sites that must serve lots of mostly static documents (such as images, streaming media, documents, and executables) to many users over a wide geographical region can benefit from using CDNs.

A CDN uses redirector servers to perform a lookup service that helps it decide which surrogate cache should satisfy a specific request. The decision is based on load estimates on the surrogate caches, customer location, and value-added services provided to customers based on their specific characteristics (such as location or domain). The main advantages of using a CDN are

- *Lower latency to the end user.* Requested documents tend to come from caches closer to the user than the originating server.
- *Reduced originating server load.* Because the originating server serves fewer requests, its bandwidth, processing, and I/O requirements shrink.
- *Increased document availability.* Several copies of the same document are cached in more than one location, making these documents available even if servers fail.
- *Reduced network traffic.* Caches located at the network's edge serve documents stored at CDN caches, which means less traffic crosses the network backbone.
- *Better "flash crowd" handling.* Special events (such as breaking news or the release of new antivirus definition files) can overwhelm certain sites by creating a traffic surge. A CDN serving part of the site content could create replicas of the data as needed to cope with demand fluctuation.

Consider a company Web site that receives 10 requests/sec. Table 1 shows a breakdown of file types and sizes.

How much bandwidth can the Web site save if a CDN served all documents (except for the dynamically generated ones)? To answer this question, we must determine how many bits per second the CDN serves instead of the Web site. Using Table 1's data and an arrival rate of 10 requests/sec, we get the following saved bandwidth:

$$10 \times (0.65 \times 18 + 0.15 \times 15 + 0.04 \times 324 + 0.03 \times 12{,}000 + 0.01 \times 1{,}336) \times 8 \times 1{,}000 = 32 \text{ MBPS}$$

In general, the saved bandwidth B is equal to

$$B = \lambda \sum_{r=1}^{R} f_r \times s_r ,$$

where λ is the average arrival rate of requests, f_r is the percentage of requests of type r, s_r is the average size of the result of requests of type r, and R is the number of types of requests served by a CDN. Obviously, Web sites must compare the cost savings in bandwidth and site infrastructure accrued from using a CDN against the cost of storing and disseminating Web documents using a CDN.

Final Remarks

With Web site caching, Web content accelerates because documents likely to be requested are maintained in the cache. This approach requires all incoming requests to pass through the cache first. Consequently, the cache server must have the capacity to handle all incoming traffic plus the cache update requests caused by cache misses.

More precisely, suppose that a site receives λ requests/sec. If the cache-object hit ratio is equal to h, then $\lambda \times (1 - h)$ requests/sec are passed on to the Web servers. If the cache must be updated for each miss, the cache must process an additional $\lambda \times (1 - h)$ requests/sec, making the total load on the cache equal to $\lambda (2 - h)$ requests/sec.

In most cases, CDNs only cache static objects. However, the growth in the number of dynamically generated Web pages calls for new techniques to cache dynamic content as well.[8]

References

1. D.A. Menascé and V.A.F. Almeida, *Scaling for E-Business: Technologies, Models, Performance, and Capacity Planning*, Prentice Hall, 2000.
2. D.A. Menascé, "Trade-Offs in Designing Web Clusters," *IEEE Internet Computing*, vol. 6, no. 5, 2002, pp. 76–80.
3. B.D. Davison, "A Web Caching Primer," *IEEE Internet Computing*, vol. 5, no. 4, 2001, pp. 38–45.
4. V.A.F. Almeida et al., "Characterizing Reference Locality in the WWW," *Proc. 4th Int'l Conf. Parallel and Distributed Information Systems (PDIS)*, IEEE CS Press, 1996, pp. 92–103.
5. D.E. Knuth, *The Art of Computer Programming: Vol. 1, Fundamental Algorithms*, Addison-Wesley, 1968.
6. L. Breslau et al., "Web Caching and Zipf-Like Distributions: Evidence and Implications," *Proc. IEEE Infocom*, IEEE Press, 1999, pp. 126–134.
7. D.A. Menascé et al., "A Hierarchical and Multiscale Approach to Analyze E-Business Workloads," *Performance Evaluation*, Elsevier Science, 2003.
8. M. Naaman, H. Garcia-Molina, and A. Paepcke, *Evaluation of Delivery Techniques for Dynamic Web Content*, tech. report, Dept. of Computer Science, Stanford Univ., 2003, http://dbpubs.stanford.edu/pub/2003-7.

Daniel A. Menascé is a professor of computer science, the co-director of the E-center for E-Business, and the director of the MS in E-Commerce program at George Mason University. He received a PhD in computer science from UCLA and authored *Capacity Planning for Web Services* and *Scaling for E-Business* (Prentice Hall, 2002 and 2000). He is a recipient of the A.A. Michelson Award from the Computer Measurement Group.