Building a Multisite Web Architecture

Optimizing components before optimizing the system as a whole can help large organizations deploy efficient, geographically redundant Web infrastructures.

Chad M. Steel
Penn State University

Many large organizations that first "came online" in the late 1990s are now facing the decision whether to upgrade their Web systems or to start anew. Given the speed with which new technologies are introduced in the Web environment, system deployment life cycles have shrunk significantly — but so have system life spans. After only a few years, an organization's Internet infrastructure is likely to need a major overhaul.

In late 2001, the systems architecture team to which I belong took on these issues for an organization that wanted to rebuild its Web infrastructure. The existing infrastructure contained multiple single points of failure, could not scale to expected usage patterns, was built on proprietary systems, and had a high management overhead. The legacy infrastructure had grown organically over the previous five years as administrators added unplanned features and functionality, and usage had grown 100-fold since the specifications were initially developed. Because of the age and condition of the legacy systems, we decided to redesign the solution from scratch to overcome the inherent limitations.

This case study describes the process our systems architecture team followed for designing and deploying the new architecture. I detail the component-selection rationale, with implementation details where allowed. Ours is just one successful approach to deploying a multisite, fully redundant Web-based system for a large organization; other reasonable and viable ways to build such a system also exist. Nonetheless, we learned several important lessons along the way — some of which contrast with conventional wisdom.

Planning the Project

A key factor differentiating this project from a typical software development

 1089-7801/02/$17.00©2002 IEEE

endeavor was the need to perform purchasing and infrastructure installation while the application architecture team was finalizing the application specifications. Due to this constraint, the systems architecture team had to adopt a process flexible enough to accommodate major changes without additional cost or delay. To that end, we borrowed heavily from the architecture tradeoff analysis method (ATAM)[1] and Hatley-Pirbhai[2] systems specification methodologies, as well as from our own experiences.

Using an increasing-refinement process (going from black box to specific component implementation details), we created the initial design, which we refined through back-propagation. We made component-level changes when possible, and changes at the physical architecture level when necessary. The initial logical architecture proved flexible enough that we did not need to make any changes at this layer.

Based on newly available information and following general system-architecture design principles, we identified several deficiencies in the previous infrastructure that drove many of our design goals.

- *Scalability*. Allow for singular (upgrades within a machine), horizontal (addition of machines within an architectural tier), and geographic (addition of another site) growth.
- *Reliability*. Provide 99.99 percent uptime for end users.
- *Extensibility*. Permit interaction with multiple database and application server platforms, and support porting of a legacy code base (C/C++, Perl, Python, and shell scripts).
- *Performance*. Support 100 million daily page views at either of two geographic locations.
- *Cost*. Maximize price and performance based on a five-year system life cycle.
- *Security*. Adhere to U.S. National Information Assurance Certification and Accreditation Process (NIACAP)[3] principles.

We used these criteria to create a physical architecture (illustrated in Figure 1) that supports all the logical architecture functionality for two geographic locations. Additionally, we used these design principles to evaluate and rank components we considered purchasing. Due to the organization's special requirements, however, we were forbidden to use open-source software on the system, which limited our scope in evaluating components.

Building the Network

The crux of the system architecture effort was translating the logical requirements into a detailed physical implementation. Based on the logical architecture and business requirements, we designed a detailed infrastructure.

When we started the project, our first decision was whether to adopt a Windows- or Unix-based solution for the environment. Given the need for extensibility, we surveyed the primary application server vendors and found the performance on Unix systems was significantly higher than the same software on Windows systems. Based on our testing, we believe that the hardware for a Windows platform could scale to the same levels as the equivalent Unix hardware, but the requirement for more CPUs for an equivalent throughput on Windows made software licensing cost prohibitive. We therefore chose to deploy a Sun Solaris-based switched, 100-Mbit-per-second Ethernet network, broken into virtual local area networks (VLANs).

To eliminate single points of failure, each network component contains two network cards cross-connected to two separate switches. This redundancy ensures that individual network component failures will not affect other components. On the front end, dual 100-Mbps active-active connections (both carrying traffic at all times) link directly to the Internet backbone at both geographic locations for a total aggregate bandwidth of 400 Mbps. Each pair of connections is then cross-linked to redundant backbone routers for upstream redundancy.

For site-to-site connectivity, we installed a point-to-point T1. We used Symmetrix Remote Data Facility (SRDF)[4] functionality to schedule automated data replication between EMC units. Considering the small number of regular changes the organization's staff would have to make after the initial load of the site content, the T1's 1.5-Mbps bandwidth seemed sufficient. Moreover, the network architecture ensures that IT staff can increase the bandwidth by setting up a virtual private network through the front-end Internet connections on demand; in fact, that is how we initially replicated the full site.

Network Infrastructure

Most of the software licensing we investigated was based on the number of CPUs, rather than the number of boxes. The cost-to-performance ratio for a Sun Solaris-based 100-Mbps Ethernet infrastructure was better than the other major Unix vendors, in part due to price drops resulting from a newly introduced product line. Additionally, we

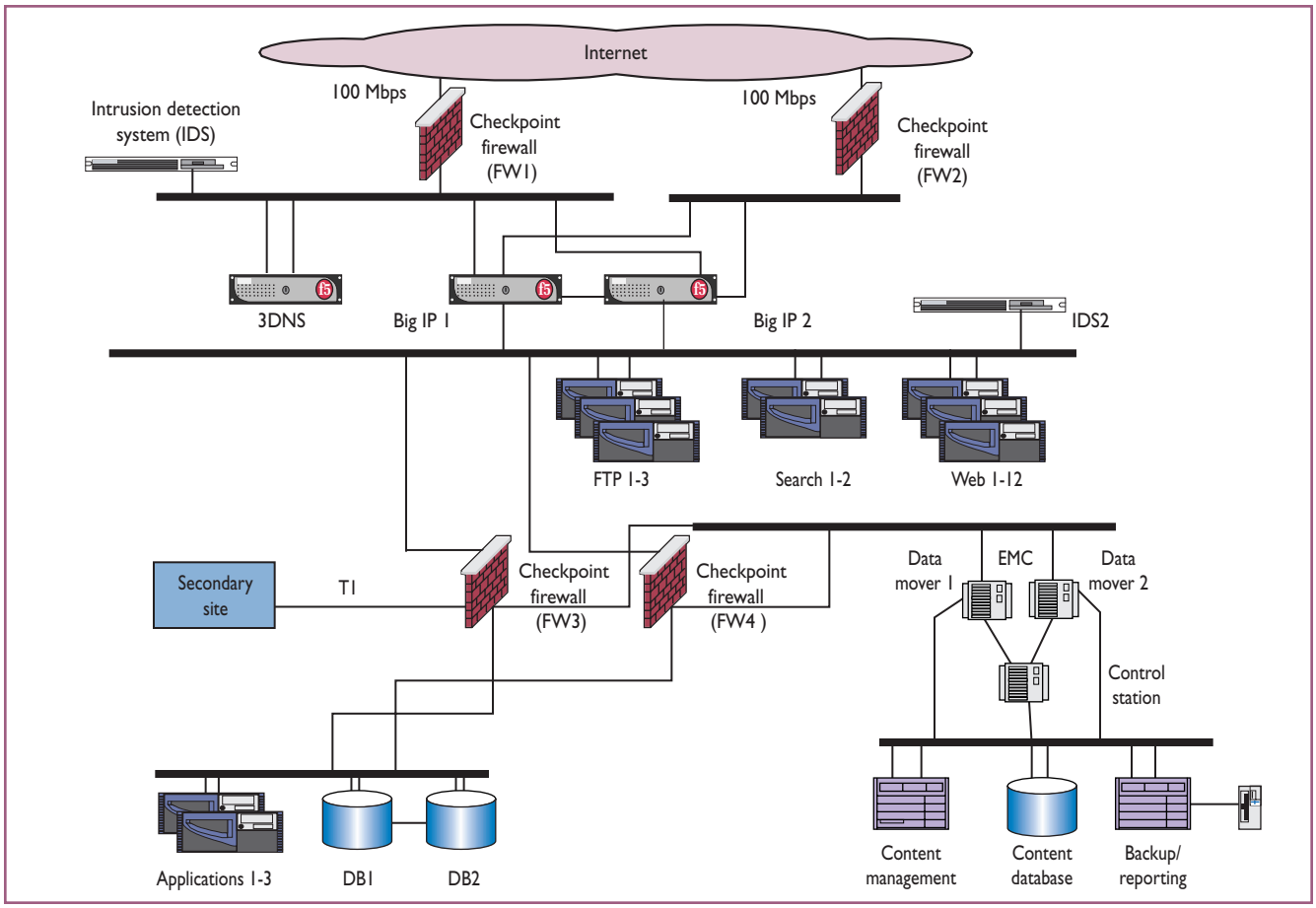

Figure 1. Physical production environment layout. The physical site architecture of the primary location contains compartmented applications with intrasite redundancy.

had access to better software support for the components we needed and more employees who were trained to use Solaris than the other major Unix vendors.

In looking at the server hardware, the main factors we considered were scalability and the price-to-performance tradeoff. We investigated hardware of various sizes — from single-rack-unit, single-CPU blade servers to standalone 64-CPU enterprise-class servers — to find the best option for this implementation. Based on the traffic patterns and expected usage (mostly front-end, static Web serving), we decided the enterprise-class servers were overkill. The single blade servers were a reasonable alternative, but the number of machines required would have necessitated higher ongoing costs for support and management, as well as additional networking hardware to implement appropriately (more switch ports, additional load-balancer workload). Given these tradeoffs, we chose to implement quad-CPU-capable, 4-rack-unit machines with memory and CPU at approximately half of capacity (2 CPUs, 2 Gbytes of RAM).

We purchased equivalently configured models for all the servers in the implementation, except the reporting server, for two primary reasons. First, it made parts-sparing (keeping spare parts on hand for replacement) and support easier because only one set of hardware was needed to replace any machine that had difficulties. Second, although a given machine might not have the optimal memory-to-CPU ratio for a given application, having equivalent machines for each tier allowed us to correct the balance between application servers, Web servers, database servers, and search servers if our initial capacity-planning estimates were off, or if new applications altered usage patterns.

We left the machines at half capacity for growth reasons. For intramachine growth, we could upgrade all of the machines in a given tier to achieve approximately twice the power for minimal cost. This allowed us to add capacity without having to add machines within a tier, eliminating additional charges for power, rack space, monitoring, and other per-machine costs.

The server used for exception reporting required

less bandwidth on the network side but significantly higher CPU and memory to process ad hoc queries and handle nightly log processing. Using the same model we implemented for the Web servers would have added complexity through the use of distributed processing. To eliminate the need for load-balancing, we decided to set up a single, more powerful machine and give it access to the final log data. We chose a mid-range server with 8 CPUs and 8 Gbytes of RAM. This also satisfied our evidentiary and security needs, which necessitated quick access to consolidated, historical log data.

Security

The system needs called for a security-in-depth model. Because security is handled across many levels of the infrastructure, I will limit the discussion to the two components of the basic setup.

Firewalls. The systems architecture team chose to use stateful inspection Checkpoint Firewall-1 appliances, which track each connection traversing all the firewall's interfaces to make sure they are valid, based on the rules we defined. Given the peak volumes expected, proxying solutions would have limited performance. Besides, the firewall boxes we chose offer robust management capabilities. For these reasons, we implemented them across the architecture, on the front-end and back-end connections as well as internally.

As Figure 1 shows, the firewalls are active-active, as are the other front-end components. The firewalls are locked down to allow only Web and FTP traffic inbound (ports 21, 80, 443) and mail traffic (SMTP) outbound; they block all other traffic at the front. We also put antispoofing rules in place on the front-end firewalls. The back-end firewalls are locked down to allow specific IP addresses from remote locations direct access to one or more boxes and to disallow outbound traffic from the site.

Intrusion detection systems. For each IDS, we installed an enhanced Network Flight Recorder (www.nfr.com) appliance to analyze and track all network activity. Our NFR appliances run on a stripped-down Linux kernel; to avoid direct targeting, they are not assigned IP addresses. We enabled port-spanning on the switches to allow the IDS to listen in promiscuous mode to all incoming and outgoing traffic on all ports.

To ensure that both internal and external traffic was monitored adequately, we placed the IDSs at two separate levels in the infrastructure. At each location, IDS1 monitors all incoming traffic before load-balancing, as well as monitoring attacks to and from the 3-Domain-Name-Service (3-DNS) box, which performs geographic load-balancing and name resolution services. We placed IDS2 behind the load-balancers so that we could monitor traffic to and from non-load-balanced IP addresses. This allows us to identify the specific box attacked, not just the load-balanced address, for later forensics.

In addition to this configuration, we followed best practices to harden operating systems and applications across the board. We performed the appropriate vulnerability assessments and penetration tests before taking the infrastructure live.[5]

Internet Software

We selected iPlanet as the Web server software for its throughput capabilities with static and dynamic pages and its ability to integrate easily with back-end application servers running Netscape server application programming interface (NSAPI) plug-ins.[6] We considered using Apache, but decided against it due to our open-source restrictions and the lack of multithread support in Apache at the time (pre-2.0).[7]

For basic FTP functionality, we used the built-in FTP software on dedicated servers. The old architecture routed all large downloads (a significant portion of the site's Internet traffic is Adobe PDF files) through the Web servers by default. We separated out the FTP servers to ensure that any compromise of an FTP server would not affect the HTTP servers (which have FTP disabled). This also allows us to limit any performance differentials to a given protocol, which makes correcting the problem quicker and more efficient.

For search functionality we chose Inktomi, primarily for its ranking technology and its ability to parse PDF files. We liked Google, but rejected it due to the inherent problems with link structure in the original site. If we had needed to go live with the legacy Web site intact, Google's algorithm would have caused suboptimal search performance due to inadequacies in the site's information architecture. Once the site underwent a separate information architecture revision, however, it would have worked just as well with Google.

Geographic Load-Balancing

To benefit full-time from the hardware implemented for the secondary production site, we set both sites to be active-active. Each location, how-

ever, was sized to handle the full peak traffic load in the event of catastrophic site failure. As an added benefit, we could then remove an entire geographic location from the load-balancing schema for scheduled maintenance and testing with no end-user impact.

We used F5's 3-DNS for geographic load-balancing (GLB) between the two locations. Traditional DNS services (we use the GLB machines for this as well) support the load-balancing algorithms by assigning one of the two locations to each request. Specifically, we used quality-of-service measurements to implement local DNS (LDNS)-persistent round-robin load-balancing. Administrators assign each LDNS server a load-balanced IP address for one of the two geographic locations, depending on how well the locations are performing. For sessioning, we maintain persistence by assigning a consistent geographic location to an LDNS. Each LDNS always has the same geographic location assigned to it unless a failure occurs for the whole location, at which point all LDNSs would be assigned the remaining location.

Along with load-balancing the two geographic locations, we used the GLB machines to load-balance each incoming 100-Mbps segment. This allowed us to treat each segment as its own geographic location for load-balancing purposes, and let us provide accurate load distribution between segments within the same data center without the added complexity of performing bonding (by combining the incoming physical connections into one logical connection) or other load distribution at the lower levels of the network stack. As an added benefit, using the GLB devices to simulate geographic load-balancing within a center allows for the removal of an entire segment for testing and upgrading while the other three segments still serve pages.

Local Load-Balancing

We installed redundant F5 Big-IP devices to perform local load-balancing (LLB). These machines provide a load-balanced IP address to the Internet and distribute incoming requests locally based on the local servers' availability and response times. The LLB machines are set to load-balance requests between the three primary server groups: Web, search, and FTP.

The LLB boxes also have secure sockets layer (SSL) accelerator cards that improve the speed of secured sessions. The boxes maintain session persistence geographically using the 3-DNS machine.

Table 1. Load-testing results.

Simultaneous virtual users	Hits/second	CPU utilization (%)	Throughput (Mbps)
500	453	1	16
1,000	810	2	32
1,500	1,177	3	45
2,000	1,511	4	62
2,500	2,011	4	80
3,000	2,556	7	96
3,500	2,905	10	110
4,000	3,304	11	128
4,500	3,751	13	145
5,000	4,236	15	160
5,500	4,776	14	173
6,000	5,246	20	191
6,500	5,647	19	208
7,000	6,071	24	228
7,500	6,474	25	238
8,000	6,818	27	250
8,500	7,040	27	262
9,000	7,326	26	270
9,500	7,482	27	275
10,000	7,428	28	278

The LLB devices provide one more function, based on their extended verification options: If checking keywords on the page turns up a change to the homepage (typical of a page-defacement attack) that invalidates the verification, the LLB box removes that machine from the active pool.

Data Storage and Replication

For primary storage in this implementation, we installed an EMC Symmetrix mass-storage device at both geographically separated data centers. Two primary factors drove our selection: ease of replication (using SRDF) and ease of management.

For replication, all content is pushed to a single partition (shared by the staging environment, where content management resides) on the mass-storage device that the content management servers can read and write to. A set of automated scripts copies updated content from this partition directly to a production partition as needed.

The production partition contains the shared content for all the Web and FTP servers, accessible through network file system (NFS) data movers in front of the mass-storage unit. To prevent page-defacement attacks, this partition is read-only from these servers. Because the FTP root is a branch of the Web root, all documents are published to a single point, which simplifies version control and

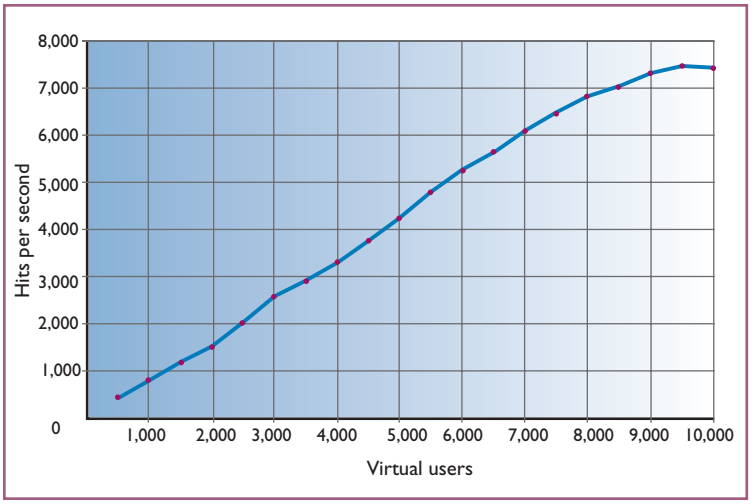

Figure 2. Hits per second progression. Hits per second increased to a peak of 7,482 — approximately 650 million hits a day.

update management. The primary mass-storage unit uses SRDF to replicate the production partition to the secondary site as needed.

Assuming properly configured local-server caching, an external storage arrangement such as this (after it is primed by traversing the entire site one time) offers performance equivalent to local subsystems without increasing complexity.

Content Management

When evaluating the organization's requirements, we determined that the content management application (CMA) should reside wholly in staging. Because the organization had an existing enterprise license, however, we needed to use Vignette's V6 Content Suite as the content management solution.

The standard Vignette V6 architecture relies on dynamic content delivery through an in-line application content server model. Given that we did not need dynamic content delivery, and that the standard architecture adds complexity and overhead, we decided to implement V6 solely in staging.

Our implementation contains the workflow and templating functions. The CMA dynamically generates a static version of each page on the site and stores it on the staging partition for use in both manual and automated testing. An automated script then moves any changes up to production after appropriate approvals by editorial, security, and testing personnel; we plan to implement more complex workflows and end-user content submission in later phases.

Considering the fairly static publishing model, even a catastrophic failure of content management would not impact end users. We therefore decided to implement the content management environment at only one location because we could not justify the additional cost of maintaining a hot backup. However, we did provision space and establish backup plans to provide a quick (within 48 hours) CMA recovery to the second geographic site as needed.

Backup, Logging, and Reporting

Separate servers at each location handle all backup, logging, and reporting. The centralized information storage on the mass storage unit minimized the need for tape backups. Log files and a database partition are regularly backed up to a central box, and we have a Jumpstart machine dedicated to quickly rebuilding the system and application software configuration on boxes on each tier. For version control, the Jumpstart is updated with patches as they are quality tested, leaving only the data to be backed up. Storing most of the data on a single shared partition significantly reduces backup complexity.

Each location runs logging and reporting individually, keeping the results separate. While this means there is no easy way to combine statistical reporting for ongoing usage, that limitation was outweighed by two savings: eliminating the need to transfer the logs for each server to the mass-storage unit and then across the wide-area network connection; and reducing the need for the additional processing power at each location to perform twice the amount of log analysis.

Webtrends is our primary tool for analyzing Web usage. We implemented the enterprise version of the software because it lets us perform cross-server tracking and logging within a geographic location. We manually collect additional metrics from networking equipment, IDSs, firewalls, and so on, but provide them to the end client only on request.

Testing the System

Before going live with the new infrastructure, we load-tested it to verify that it met contractual performance requirements and to remove any remaining bottlenecks.

Testing entailed simulating up to 10,000 simultaneous 28.8-Kbps modem users (the most common usage speed from the previous year). We set the virtual users to travel specific paths of varying lengths through the site and download forms of various sizes. The specific paths and forms were not random, but were generated to approximate the highest usage patterns from data collected on

the legacy site. We used actual form sizes to reduce inflation of numbers due to inaccurate simulation from smaller (or even larger) page sizes.[8]

Results

Table 1 shows the combined results for both sites. The primary measures we collected were hits per second, CPU utilization, and throughput. (Earlier testing revealed that both memory and disk utilization had extremely minimal impact on performance, so we did not monitor them further.) The test results showed that performance was bound by the bandwidth more than by the architecture.

As Figure 2 (previous page) illustrates, the primary resources scaled linearly until they reached the highest utilization numbers — indicating that our design should continue to scale with expected usage. Figure 3 shows how contention issues started to affect performance once combined bandwidth usage passed 270 Mbps for the four 100-Mbps links.

CPU utilization remained on a linear progression as well (Figure 4). We had substantially more CPU power at our disposal than originally expected, perhaps due in part to optimizations we performed on the iPlanet servers (tweaking options individually during earlier component load testing). Based on the load-testing results, we put migration plans in place to extend the bandwidth of the incoming connections to a gigabit feed if usage increased enough to mandate it.

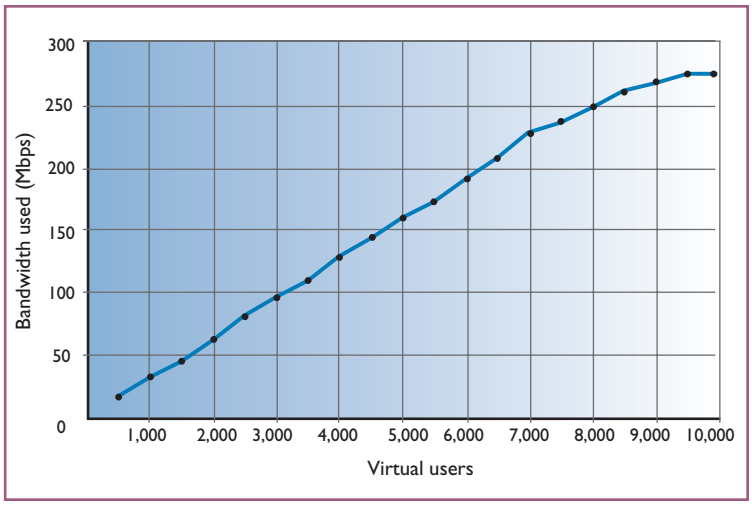

Figure 3. Throughput progression. Total bandwidth for the two locations peaked at 278 Mbps for the four connections.

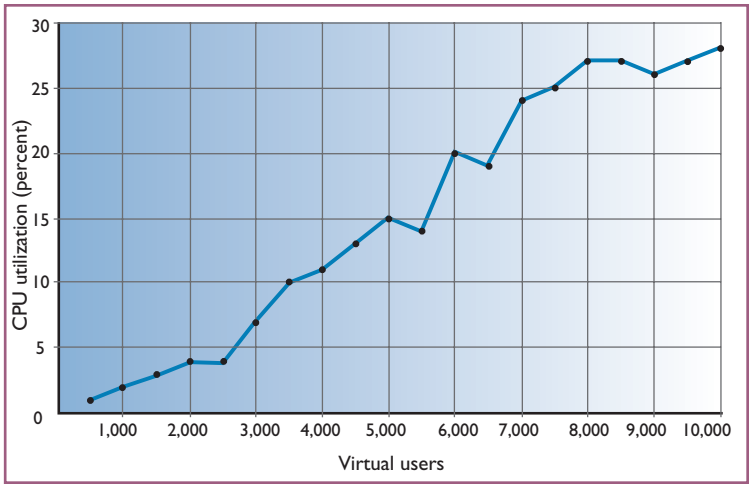

Figure 4. Average CPU utilization progression. Average CPU utilization peaked at 28 percent, indicating that this was not a choke point for performance.

Lessons Learned

The following list summarizes our top seven lessons we learned in deploying this complex, multilocation Web-based system.

Component load-testing and optimization is critical. We improved performance by 3 to 4 times on several individual components by load-testing them on individual servers with their specific applications. Application and network-tuning issues that might not have become apparent during the full testing were more prominent during the individual tests. Adding one component at a time to the testing configuration enabled us to isolate performance problems and optimize individual performance before system performance.

Ensure communications between groups. One of our primary challenges was to maintain open communications between groups. Although this is not an uncommon issue in such projects, it becomes critical in optimizing a system architecture because an application architecture decision can have a tenfold impact on overall system performance. Good communication helped us resolve at least one serious problem: a particular software component was performing noncached lookups from the database servers, creating an application-level bottleneck. By establishing cross-functional troubleshooting teams, we saved significant time in identifying the appropriate group to fix any issue that came up, and finger-pointing (which often results from poor communications) was nonexistent.

Past usage doesn't describe future usage. Once the architecture went into production, new usage patterns were significantly different from those shown in the historical data. This was partly due to a more efficient design, but factors like a more prominent search box and changes in the overall

economic landscape altered public interest and usage patterns as well.

Don't forget the administrative interfaces. Although I skimmed over the topic in this article, the security infrastructure presented one of the main challenges to the implementation. The largest problem was integrating the management and administration interfaces into a comprehensive architecture. For example, the IDS needs an interface to send secure alerts, the backup subsystems need their own LAN, and the monitoring software needs primary and backup solutions for sending data.

Put an IDS after load-balancing. If your only IDS is in front of the load-balancers, it is difficult to track incoming requests to an individual machine. When determining where a successful attack hit, it saves time and energy to be able to map to an individual box. Additionally, a front-end IDS might not catch traffic between already compromised internal boxes.

Talk to the component designers when anomalies occur. At one point, performance dropped sharply on the Web server, and we were unable to diagnose it locally. When we engaged Sun's support staff, we determined that the wait queue length was fixed at a suboptimal number for our implementation. Changing that single parameter made a drastic impact on overall performance.

Don't be afraid to go against established implementation models. To suit the mandates of the business requirements, we had to change several details of the implementation (not the requirement!), although that went against traditional vendor deployment models. For example, V6 is not usually used in a "dynamic page generation in staging pushed to static production pages" model, but that configuration gave us robust content-management capabilities without the overhead or added complexity of dynamic generation. Also, using the LDNS to load-balance local segments allowed us to use all four segments in active-active mode without a complex network configuration.

Conclusion

The site is now successfully live, and it has won accolades from both internal groups and the media for its improved design and performance. Implementing this system allowed us to apply existing methodologies to Web architecture design.

There are specific methodologies for developing Web sites and traditional Web-based applications. Additionally, there are many useful systems architecture methodologies available. Based on lessons learned from large Web deployments and wisdom distilled from existing methodologies, the next challenge will be to create a comprehensive methodology specifically for Web system architecture.

References

1. P. Clements, R. Kazman, and M. Klein, *Evaluating Software Architectures: Methods and Case Studies*, Addison-Wesley, Boston, Mass., 2001.
2. D. Hatley and I. Pirbhai, *Strategies for Real-Time System Specification*, Dorset House, New York, 1987.
3. U.S. Govt. Standard NSTISSI 1000, *National Information Assurance Certification and Accreditation Process*, U.S. National Institute of Standards and Technology, Gaithersburg, MD, Apr. 2000.
4. "Symmetrix Remote Data Facility," tech. note, EMC Corp., Hopkinton, Mass., 2001.
5. P. Herzog, "Open Source Security Testing Methodology Manual," GNU Public License Document, 2002; available at http://uk.osstmm.org/osstmm.en.2.0.zip.
6. *iPlanet Web Server, Enterprise Edition Administrator's Guide*, Sun Microsystems, Palo Alto, Calif., 2001; available at http://docs.sun.com/source/816-5691-10/.
7. B. Weiner, "iPlanet Web Server, Enterprise Edition 4.0 and Stronghold 2.4.2 Performance Comparison," Mindcraft Corp., London, 2000.
8. D. Yates, V. Almeida, and J. Almeida, "On the Interaction between an Operating System and a Web Server," tech. report, CS97-012, Boston Univ., 1997.

Chad M. Steel is the director of systems integration and security services for a Fortune 500 consulting organization and an adjunct professor in software engineering at Penn State University. He holds a BS and an MS in computer engineering from Villanova University. His research interests include Internet security and Web architectures.

Readers can contact the author at csteel@ieee.org.