September, 2002

Globally Distributed Content Delivery

John Dilley
Bruce Maggs
Jay Parikh
Harald Prokop
Ramesh Sitaraman, *University of Massachusetts - Amherst*, et al.

Globally Distributed Content Delivery

Using more than 12,000 servers in over 1,000 networks, Akamai's distributed content delivery system fights service bottlenecks and shutdowns by delivering content from the Internet's edge.

John Dilley, Bruce Maggs, Jay Parikh, Harald Prokop, Ramesh Sitaraman, and Bill Weihl
Akamai Technologies

As Web sites become popular, they're increasingly vulnerable to the flash crowd problem, in which request load overwhelms some aspect of the site's infrastructure, such as the front-end Web server, network equipment, or bandwidth, or (in more advanced sites) the back-end transaction-processing infrastructure. The resulting overload can crash a site or cause unusually high response times — both of which can translate into lost revenue or negative customer attitudes toward a product or brand.

Our company, Akamai Technologies, evolved out of an MIT research effort aimed at solving the flash crowd problem (www.akamai.com/en/html/about/history.html). Our approach is based on the observation that serving Web content from a single location can present serious problems for site scalability, reliability, and performance. We thus devised a system to serve requests from a variable number of surrogate origin servers at the network edge.[1] By caching content at the Internet's edge, we reduce demand on the site's infrastructure and provide faster service for users, whose content comes from nearby servers.

When we launched the Akamai system in early 1999, it initially delivered only Web objects (images and documents). It has since evolved to distribute dynamically generated pages and even applications to the network's edge, providing customers with on-demand bandwidth and compute capacity. This reduces content providers' infrastructure requirements, and lets them deploy or expand services more quickly and easily. Our current system has more than 12,000 servers in over 1,000 networks. Operating servers in many locations poses many technical challenges, including how to direct user requests to appropriate servers, how to handle failures, how to monitor and control the servers, and how to update software across the sys-

 1089-7801/02/$17.00 ©2002 IEEE

tem. Here, we describe our system and how we've managed these challenges.

Existing Approaches

Researchers have explored several approaches to delivering content in a scalable and reliable way. Local clustering can improve fault-tolerance and scalability. If the data center or the ISP providing connectivity fails, however, the entire cluster is inaccessible to users. To solve this problem, sites can offer mirroring (deploying clusters in a few locations) and multihoming (using multiple ISPs to connect to the Internet). Clustering, mirroring, and multihoming are common approaches for sites with stringent reliability and scalability needs. These methods do not solve all connectivity problems, however, and they do introduce new ones:

- It is difficult to scale clusters to thousands of servers.
- With multihoming, the underlying network protocols — in particular the border gateway protocol (BGP)[2] — do not converge quickly to new routes when connections fail.
- Mirroring requires synchronizing the site among the mirrors, which can be difficult.

In all three cases, excess capacity is required: With clustering, there must be enough servers at each location to handle peak loads (which can be an order of magnitude above average loads); with multihoming, each connection must be able to carry all the traffic; and with mirroring, each mirror must be able to carry the entire load. Each of these solutions thus entails a considerable cost, which could more than double a site's initial infrastructure expense and ongoing operation costs.

The Internet is a complex fabric of networks. Congestion and failures occur at many different places, including

- the "first mile" (which is partially addressed by multihoming the origin server),
- the backbones,
- peering points between network service providers, and
- the "last mile" to the user.

Deploying independent proxy caches throughout the Internet can address some of these bottlenecks. Transit ISPs and end-user organizations have installed proxy caches to reduce latency and bandwidth requirements by serving users directly from a previously requested content cache. However, Web proxy cache hit rates tend to be low — 25 to 40 percent — in part because Web sites are using more dynamic content. As a result, proxy caches have had limited success in improving Web sites' scalability, reliability, and performance.

Akamai works closely with content providers to develop features that improve service for their Web sites and to deliver more content from the network edge. For example, features such as authorization, control over content invalidation, and dynamic content assembly let us deliver content that would otherwise be uncacheable. Although ISP caches could include similar features, to be useful they would have to standardize the features and their implementation across most cache vendors and deployments. Until such a feature is widely deployed, content providers have little incentive to use it. Because Akamai controls both its network and software, we can develop and deploy features quickly.

Akamai's Network Infrastructure

Akamai's infrastructure handles flash crowds by allocating more servers to sites experiencing high load, while serving all clients from nearby servers. The system directs client requests to the nearest available server likely to have the requested content. It determines this as follows:

- *Nearest* is a function of network topology and dynamic link characteristics: A server with a lower round-trip time is considered nearer than one with a higher round-trip time. Likewise, a server with low packet loss to the client is nearer than one with high packet loss.
- *Available* is a function of load and network bandwidth: A server carrying too much load or a data center serving near its bandwidth capacity is unavailable to serve more clients.
- *Likely* is a function of which servers carry the content for each customer in a data center: If all servers served all the content — by round-robin DNS, for example — then the servers' disk and memory resources would be consumed by the most popular set of objects.

In the latter case, an Akamai site might hold a dozen or more servers within any data center; the system distributes content to the minimum number of servers at each site to maximize system resources within the site.

Automatic Network Control

The direction of requests to content servers is referred to as mapping. Akamai's mapping tech-

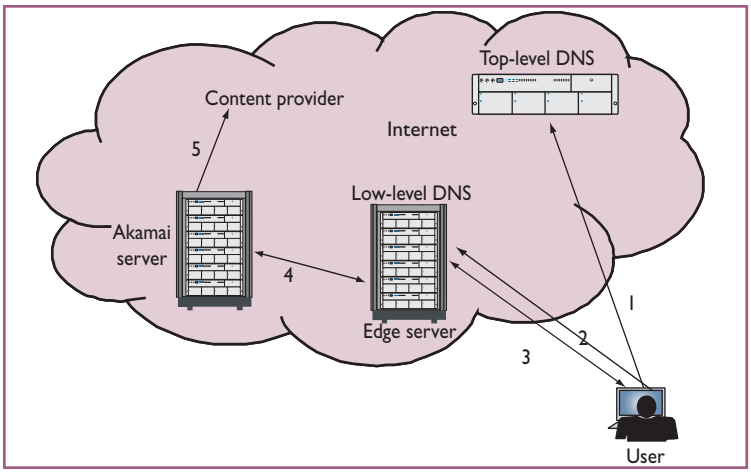

Figure 1. Client HTTP content request. Once DNS resolves the edge server's name (steps 1 and 2), the client request is issued to the edge server (step 3), which then requests content from the appropriate source (step 4 or 5), satisfies the request, and logs its completion.

nology uses a dynamic, fault-tolerant DNS system. The mapping system resolves a hostname based on the service requested, user location, and network status; it also uses DNS for network load-balancing.

The "DNS Resolution" sidebar describes the standard DNS resolution process for an Akamai edge server name, corresponding to steps 1 and 2 in Figure 1. In step 3, the client makes an HTTP request to the edge server, which then retrieves the content by requesting it from either another Akamai server (step 4) or the content provider's server (step 5). The server then returns the requested information to the client and logs the request's completion.

Akamai name servers resolve host names to IP addresses by mapping requests to a server using some or all of the following criteria.

- *Service requested*. The server must be able to satisfy the request. The name server must not direct a request for a QuickTime media stream to a server that handles only HTTP.
- *Server health*. The content server must be up and running without errors.
- *Server load*. The server must operate under a certain load threshold and thus be available for additional requests. The load measure typically includes the target server's CPU, disk, and network utilization.
- *Network condition*. The client must be able to reach the server with minimal packet loss, and the server's data center must have sufficient bandwidth to handle additional network requests.
- *Client location*. The server must be close to the client in terms of measures such as network round trip time.
- *Content requested*. The server must be likely to have the content, according to Akamai's consistent hashing algorithm.

Internet routers use BGP messages to exchange network reachability information among BGP systems and compute the best routing path among the Internet's autonomous systems.[2] Akamai agents communicate with certain border routers as peers; the mapping system uses the resulting BGP information to determine network topology. The number of hops between autonomous systems is a coarse but useful measure of network distance. The mapping system combines this information with live network statistics — such as traceroute data[3] — to provide a detailed, dynamic view of network structure and quality measures for different mappings. Implementing this mapping system on a global scale involves several challenges, as we discuss later.

Network Monitoring

Our DNS-based load balancing system continuously monitors the state of services, and their servers and networks. Each of the content servers — for the HTTP, HTTPS, and streaming protocols — frequently reports its load to a monitoring application, which aggregates and publishes load reports to the local DNS server. That DNS server then determines which IP addresses (two or more) to return when resolving DNS names. If a server's load exceeds a certain threshold, the DNS server simultaneously assigns some of the server's allocated content to additional servers. If the load exceeds another threshold, the server's IP address is no longer available to clients. The server can thus shed a fraction of its load when it is experiencing moderate to high load. The monitoring system also transmits data center load to the top-level DNS resolver to direct traffic away from overloaded data centers.

To monitor the entire system's health end-to-end, Akamai uses agents that simulate end-user behavior by downloading Web objects and measuring their failure rates and download times. Akamai uses this information to monitor overall system performance and to automatically detect and suspend problematic data centers or servers.

In addition to load-balancing metrics, Akamai's monitoring system provides centralized reporting on content service for each customer and content server. This information is the basis of Akamai's real-time customer traffic analyzer application. The information is useful for network operational and diagnostic purposes, and provides real-time

DNS Resolution

Akamai edge servers are located using a DNS name, such as a7.g.akamai.net. A DNS resolver resolves this name in the standard manner, from right to left, querying DNS name servers until IP addresses for the host a7 in the domain .g.akamai.net are returned.

Each name resolution associates a "time to live" (TTL) with the resolution, which proceeds as follows:

1. The resolver chooses a root name server and asks it to resolve the name a1.g.akamai.net. The root name server does not itself resolve the name; instead, it sends a domain delegation response with IP addresses of the name servers that handle .net domain requests.
2. The resolver then queries the .net name servers, which return a domain delegation (NS records) for .akamai.net. These are the Akamai top-level name servers (top-level DNS in Figure 1).
3. Next, the resolver queries an Akamai TL DNS server, which returns a domain delegation for .g.akamai.net to low-level Akamai name servers (LL NS in Figure 1) with a TTL of about one hour. The low-level name servers selected correspond to (and are in the same location as) the available edge servers that are closest to the requesting user.
4. Finally, the resolver queries an Akamai low-level DNS server, which returns the IP addresses of servers available to satisfy the request. This resolution has a short TTL (several seconds to one minute), which encourages frequent refreshes of the DNS resolution and allows Akamai to direct requests to other locations or servers as conditions change.

A resolver is preconfigured to know the Internet root name servers' IP addresses, which are the starting points for a DNS resolution if the resolver lacks required information in its cache. If the resolver has valid IP addresses of the .net name server, it skips step 1; if it has cached IP addresses of the g.akamai.net name servers, it skips steps 1 through 3.

The resolution process is the same for any DNS name. Akamai name-resolution differs, however, in how its name servers behave.

access to an array of service parameters organized as a database. The application's SQL-like interface supports ad-hoc queries against live and historic data, which lets the operations staff locate the busiest customer, the server using the most memory or disk space, or the switch or data center closest to its bandwidth limit.

Network Services

Akamai servers deliver several types of content: static and dynamic content over HTTP and HTTPS, and streaming audio and video over the three streaming protocols described below.

Static Content

Static Web content consists of HTML pages, embedded images, executables, PDF documents, and so on. Akamai's content servers use content type to apply lifetime and other features to static documents, which have varying cacheability and can have special service requirements.

Lifetimes, for example, can vary from zero seconds, where the edge server validates the object with the origin server on each request, to infinite, where the content server never checks the object's consistency. Lifetime values for Akamai edge servers can also differ from downstream proxy servers and end users.

Special features might include the ability to serve secure content over the HTTPS protocol, support alternate content and transfer encodings, handle cookies, and so on. Akamai controls features on behalf of each customer using a metadata facility that describes which features to apply by customer, content type, and other criteria.

Dynamic Content

Today's Web sites depend heavily on dynamic content generation to offer end users rich and captivating material. As we noted earlier, however, proxy caches cannot typically cache dynamic content. A proxy cache could not, for example, handle a largely static Web page if it contained an advertisement that changed according to each user's profile.

To deal with this, we use Edge Side Includes technology (www.esi.org), which assembles dynamic content on edge servers. ESI is similar to server-side include languages, but adds fault-tolerance features (for when the origin server is unavailable) and integrates an Extensible Stylesheet Language Transformation (XSLT) engine to process XML data. Using ESI lets a content provider break a dynamic page into fragments with independent cacheability properties. These fragments are maintained as separate objects in the edge server's cache and are dynamically assembled into Web pages in response to user requests.

The ability to assemble dynamic pages from individual page fragments means that the server must fetch only noncacheable or expired fragments from the origin Web site; this reduces both the load on the site's content generation infrastructure and the data that the edge server must retrieve from the central origin server. ESI reduced bandwidth

requirements for dynamic content by 95 to 99 percent across a range of dynamic sites we studied, including portals and financial sites. The resulting reduction in central infrastructure offers content providers significant savings.

Streaming Media

Akamai's streaming network supports live and on-demand media in the three major formats — Microsoft Windows Media, Real, and Apple's QuickTime. While building a streaming delivery network presents some technical issues that are similar to those of a Web delivery network, there are also significant additional challenges.

First, the content provider typically captures and encodes a live stream and sends it to an entry-point server in the Akamai network. Given our principle of removing all single points of failure, we must have mechanisms that will react quickly to a failed entry-point server. Specifically, another entry-point server must pick up the live stream quickly enough that end users detect no interruption in the stream.

The stream is delivered from the entry-point server to multiple edge servers, which in turn serve the content to end users. Media packet delivery from the entry-point to the edge servers must be resilient to network failures and packet loss, and thus the entry point server must route packet flows around congested and failed links to reach the edge server. Further, the entry point and edge servers must deliver packets without significant delay or jitter because a late or out-of-order packet is useless in the playback. When necessary, Akamai uses information dispersal techniques that let the entry point server send data on multiple redundant paths, which lets the edge server construct a clean copy of the stream even when some paths are down or lossy.

Typically, a content provider uploads an on-demand clip into an Akamai content storage facility. We distribute the storage facility over many data centers and automatically replicate the uploaded clip to a subset of the centers. An edge server that receives a stream request downloads the content from its optimal storage location and caches it locally while serving the request.

Technical Challenges

Constructing a global network poses many nontechnical challenges, including deploying network equipment and server hardware, establishing good working relationships with network providers, controlling operational expenses, and acquiring and supporting customers. While these challenges are significant, our focus here is on challenges related to designing, building, and operating the system itself.

System Scalability

Akamai's network must scale to support many geographically distributed servers, and many customers with differing needs. This presents the following challenges.

- Monitoring and controlling tens of thousands of widely distributed servers, while keeping monitoring bandwidth to a minimum.
- Monitoring network conditions across and between thousands of locations, aggregating that information, and using it to generate new maps every few seconds. Success here depends on minimizing the overhead added to DNS to avoid long DNS lookup times; this lets us perform the calculations required to identify the optimal server off-line, rather than making the user wait.
- Dealing gracefully with incomplete and out-of-date information. This requires careful design and iterative algorithm tuning.
- Reacting quickly to changing network conditions and changing workloads.
- Measuring Internet conditions at a fine enough granularity to attain high-probability estimates of end user performance.
- Managing, provisioning, and solving problems for numerous customers with varying needs, varying workloads, and varying amounts of content.
- Isolating customers so that they are incapable of negatively affecting each other.
- Ensuring data integrity over many terabytes of storage across the network. Because low-level (file system or disk) checks are inadequate to protect against possible errors — including those caused by operators and software bugs — we also perform end-to-end checks.
- Collecting logs with information about user requests, processing these logs (for billing), and delivering accurate, timely billing information to customers.

To meet the challenges of monitoring and controlling content servers, Akamai developed a distributed monitoring service that is resilient to temporary loss of information. To solve problems for customers, Akamai has customer-focused teams that diagnose problems and provide billing services.

System Reliability

A distributed system offers many opportunities for fault-tolerance, but it also introduces many components that could fail. This is partly a problem of scale: System failures must be detected and corrected across multiple software platforms. It's also inevitable that computer hardware will age, particularly disks and other moving parts. As a large network's equipment ages, a fraction of the devices will fail each month. Network operations staff must detect failure reliably and remove units from service quickly so they can be shipped to a warehouse and perhaps repaired and returned to service.

Akamai's monitoring and mapping software ensures that server or network failures do not affect end users, and the DNS system detects failures quickly and immediately hands out new IP addresses. For customers still using cached DNS answers, two mechanisms help prevent denial of service: the DNS resolution can return multiple IP addresses, so that the client can try another address; or a live server at the site can assume the failed server's IP address. To prevent problems when network failures make sites unreachable (due to router and switch problems, for example), the top-level (TL) DNS will identify local DNS servers at different sites to ensure that clients can reach a live DNS server.

Again, to prevent outages, we must carefully avoid single points of failure throughout the system. Our edge servers provide massive replication for content delivery and DNS request processing. We also avoid single failure points by replicating monitoring and control mechanisms.

Finally, we must detect and repair software flaws. A key challenge in Web content service is that client requests and server responses are not static — new request and response headers regularly appear as vendors enhance their browsers and servers, and edge servers must interpret these changes correctly and efficiently. Because such enhancements can come at any time, it's infeasible to test new versions on all relevant browser and content-server versions; testing content server features with the top browser and content server configurations alone creates a very complex testing matrix. To address this challenge, we created a test tool that directs a copy of a live server's traffic to a test version of our software without affecting content service. This real-world traffic enables us to find problems before software is deployed to the live network.

Software Deployment and Platform Management

In addition to expanding the network's size and scope, software must evolve with new features for customers, improved performance, and better operational and monitoring capabilities. To meet this challenge, we must deploy high-quality software to servers on many networks, sometimes quickly to accommodate a rapid time-to-market business model.

We cannot upgrade software on the entire network atomically. If nothing else, prudence dictates that we deploy new network software in stages so that we can detect problems before they have significant impact. Also, it's unlikely that all edge servers (or even all networks) will be available at the same time. Inevitably, we'll miss some servers and have to update them at a later time. As a result, it's more the rule than the exception to have two versions of a software component live on the network at the same time. Given this, we must write components so that different versions can coexist and carefully manage changes to component interfaces. Network operations must continually monitor the network and suspend any incorrectly configured servers.

The servers in the Akamai network run on Linux and Windows operating systems. Operating multiple OS platforms and services requires a monitoring platform and tools that run across those platforms and have access to servers' service delivery parameters. This information is required for load balancing (local and global) as well as to support problem diagnosis for operations and customer care. Finally, running multiple platforms and applications requires expertise in all supported platforms and servers.

Content Visibility and Control

Akamai's network distributes and serves content for providers, who must retain control over their content as we serve it at the edge, and see real-time information about what content is served to whom and when. Providing this visibility and content control offers challenges in cache consistency, lifetime and integrity control, and several other areas.

Cache consistency. When objects that the edge servers deliver are cacheable, we must address the consistency of cached content; when they are uncacheable, high-performance delivery is a challenge.

To address *cacheable-object consistency*, content providers often use established techniques, such as applying a "time to live" (TTL) to objects. Some objects might be cacheable forever, or at least until they are explicitly removed by a cache control utility (for more on this, see the "Lifetime Control" sec-

Related Work in Distributed and Fault-Tolerant Systems

Akamai system technology builds on years of research and development in distributed and fault-tolerant systems. Many other parts of the Internet — such as the network routing fabric, the DNS, email, and the Web itself — rely on some communication and coordination among decentralized asynchronous components. In contrast, the Akamai system uses logically centralized (but distributed and replicated) administration and control to map requests to servers and to manage the entire system. Other systems also use logically centralized control. The Autonet,[1] for example, uses a centralized algorithm to recompute and distribute routing tables when the network topology changes. In contrast to systems like the Autonet, which recomputes routes only in response to failures and component additions, the Akamai system remaps requests continuously based on service health and load, and the network connection quality among components and between components and end users.

Web-Based Caching

Distributed systems research also has a long history of distributed data systems work on databases,[2] file systems,[3] and caches.[4] The Web offers two significant advantages here: data is primarily read-only, and the users are typically people, which means they are more tolerant of data inconsistencies than programs are. This makes keeping caches consistent with data sources relatively easy compared to similar problems in distributed databases and file systems.

The Web has long used caching, first in browsers and then in forward proxies. Recent research, however, suggests that corporate and ISP Web caching use is not significant.[5] Web caching's lack of success relative to other types of caching might be because Web proxy caches are not closely coordinated with the data sources; caching software is built and administered by organizations distinct from those that provide the data. In contrast, Akamai maintains a direct relationship with content providers, which drives innovation and allows otherwise uncacheable content to be served from the edge.

Update and Management Tools

Several package and updating management tools address software update and management in large distributed computing environments. Depot[6] achieves reliable system update using modular package and internal consistency verification. Depot relies primarily on a shared global file system for secure data distribution. Akamai, however, uses public-key techniques to deliver updates over HTTP, and achieves performance and scalability by using the Akamai network itself for the delivery.

References

1. M. Schroeder et al., "Autonet: A High-Speed, Self-Configuring Local Area Network Using Point-to-Point Links," *IEEE J. Selected Areas in Comm.*, vol. 9, no. 8, Oct. 1991, pp. 1318-1335.
2. P. Bernstein, V. Hadzilacos, and N. Goodman, *Concurrency Control and Recovery in Database Systems*, Addison-Wesley, Reading Mass., 1987.
3. M. Satyanarayanan, "A Survey of Distributed File Systems," *Annual Rev. of Computer Science*, Annual Reviews, Inc., Palo Alto, Calif., 1989, pp. 447-459.
4. B. Davidson, "A Survey of Proxy Cache Evaluation Techniques," *Proc. 4th Int'l Web Caching Workshop*, 1999; available at http://citeseer.nj.nec.com/davison99survey.html.
5. S. Gadde, J. Chase, and M. Rabinovich, "Web Caching and Content Distribution: A View From the Interior," *Computer Comm.*, vol. 24, no. 2, 2001, pp. 222-231.
6. W. Colyer and W. Wong, "Depot: A Tool for Managing Software Environments," *Large Installation System Administration (LISA) VI Proc.*, Usenix Assoc., Berkeley, Calif., 1992, pp. 153-162.

tion). Another approach is to use a different URL for each object version. In addition to using a unique query string for this purpose, we let customers place a version or generation number in the URL. Versioned objects typically have infinite TTLs.

To improve *uncacheable objects'* performance, we introduce an edge server between the client and origin to split the client's TCP connection into two separate connections – one from the client to the edge server and one from the edge server to the origin. Contrary to intuition, splitting the connection can deliver faster responses in some cases because the edge server can react to packet loss more quickly than the origin server, improving the connection's bandwidth-delay product. We also map clients to edge servers that have low congestion and packet loss. Furthermore, the edge server can accept the origin server's response faster than the client could, and can serve it from memory at the client's pace. This frees up origin server resources to serve subsequent requests, reducing origin site demand even for uncacheable content. Finally, the edge server can maintain much longer persistent connections with the client than can an origin server; the origin need only maintain connections with relatively few Akamai edge servers.

Lifetime control. In some cases, the edge server must remove certain objects from all servers on demand. This might be in response to a request from an Akamai customer (the content's provider), or initiated by an interface that lets content publishing systems schedule invalidations when content changes. Because most Web objects change infrequently,[4] heuristic caching policies in Web proxies typically hold copies long after they change. Akamai's edge servers support on-demand purges for changed or otherwise invalid content.

Authentication and authorization. When serving

protected content, edge servers must either contain authorization features or relay authentication tokens to the origin server for authorization. In the latter case, the edge server must be careful not to evict the protected content on a request authorization failure. Akamai lets content providers authorize every user request from their own site by passing request headers from our edge servers to their content servers prior to serving each client request. Akamai edge servers can also process authorization tokens that the origin server attaches to the request, thereby avoiding a round trip to the origin server on each request.

Integrity control. A server must ensure that each client request receives the correct response, and also detect when origin servers issue incomplete responses and avoid caching those responses. Edge servers can contain content from many customer origin servers, and it's imperative that they not serve content to the wrong customer — regardless of the content's name or how clients access it. Furthermore, a server should detect when cached objects become corrupted (due to disk failure, for example) and refetch them if they do. In Akamai's system, we have built a content integrity check feature into our software; prior to serving each block of a response, the server double checks that the content is associated with the request. This protects the edge server from serving content that was corrupted on disk or confused in memory due to a software error.

Visibility into access patterns. Customers want to see detailed content-access logs. To offer this, we aggregate individual server logs and extract relevant entries for each customer. Log delivery and aggregation involves a significant data flow, however, and collecting and processing all the logs can take time. Some content providers also want real-time delivery information about their site. In this case, we focus on giving customers content delivery rates and client locations, rather than full log details.

Billing. Revenue from the content providers we serve supports our network operation. Billing requires aggregating information from each server and processing it to reduce the information volume from billions of log lines to a summary of a month's worth of content access. Scaling back-end billing infrastructure is as important as scaling client-facing servers: As the number of servers and customers grows, so too does the amount of log information we must process.

Conclusion

Current work at Akamai is focused on running applications at the network's edge, bringing the benefits of wide area content delivery into the application space. Running applications on a globally distributed network of computers provides many of the same advantages as simple content delivery: capacity on demand, cost-effective use of shared resources, ability to respond to users without communicating over long distances, and so on. At the same time, it poses many new and interesting challenges. For example, customers need visibility into the behavior of their running applications, even though the machines that run a given application might change from day to day or even minute to minute. Similarly, applications must be sandboxed to ensure that one customer's application does not interfere with that of another customer. Also, applications typically need access to data, which must be distributed in some fashion along with the applications. Many of these problems are quite difficult, if not impossible, to solve in their full generality. The challenge is to identify design patterns that provide useful and cost-effective solutions.

Acknowledgments

We are grateful for comments and suggestions from Julia Austin, Chris Joerg, Hal Jordy, David Judson, Marty Kagan, Eisar Lipkovitz, Daniel Stodolsky, Perry Stoll, and the anonymous reviewers.

References

1. D. Karger et al., "Consistent Hashing and Random Trees: Distributed Caching Protocols for Relieving Hot Spots on the World Wide Web," *Proc. 29th Annual ACM Symp. Theory of Computing*, ACM Press, New York, 1997, pp. 654-663.
2. Y. Rekhter and T. Li, *A Border Gateway Protocol 4*, Internet Eng. Task Force RFC 1771, Mar. 1995; available at www.ietf.org/rfc/rfc1771.txt.
3. G. Malkin, *Traceroute Using an IP Option*, IETF RFC 1393, Jan. 1993; available at www.ietf.org/rfc/rfc1393.txt.
4. F. Douglis et al., "Rate of Change and other Metrics: A Live Study of the World Wide Web," *Symp. Internet Technology and Systems*, Usenix Assoc., Berkeley, Calif., 1997.

John Dilley is a principal architect at Akamai Technologies. He helped develop the proxy software that delivers HTTP content, managed the development team, and is now building next-generation proxy software. His research interests include distributed systems design and performance. Dilley received a BS in mathematics and a BS and MS in computer science from Purdue University. He is a member of

the IEEE Computer Society.

Bruce Maggs is an associate professor of computer science at Carnegie Mellon University and vice president of research at Akamai Technology, which he helped launch. He received a BS, MS, and PhD in computer science from the Massachusetts Institute of Technology.

Jay Parikh is a director of engineering at Akamai Technologies. He managed the development and launch of Akamai services, including ESI and FirstPoint, and currently manages the team building Akamai's next-generation Edge Computing distributed application services. He received a BS in mechanical engineering from Virginia Tech.

Harald Prokop is a senior software engineer at Akamai Technologies. He joined Akamai in 1999, and now leads the design and development of the mapping system. His research interests include high-performance computing, parallel and distributed computing, and memory-efficient algorithms. Prokop received an MS in computer science from MIT.

Ramesh Sitaraman is a principal architect at Akamai Technologies, where he heads Akamai's product and services performance group. He is also an associate professor of computer science at the University of Massachusetts, Amherst. His research interests include parallel and distributed architectures and algorithms, and he is associate editor of the *IEEE Transactions on Parallel and Distributed Systems*. Sitaraman received a PhD in computer science from Princeton University.

Bill Weihl is chief architect for edge services at Akamai Technologies. Prior to joining Akamai in 1999, he was a professor at MIT and a research scientist at Digital's Systems Research Center. His research interests include distributed and parallel computing, programming languages, fault-tolerance, and computer architecture. He is a member of ACM and the IEEE.

Readers can contact the authors at {jad, bmm, jay, prokop, ramesh, bweihl}@akamai.com.