

EdgeComputing: Extending Enterprise Applications to the Edge of the Internet

Andy Davis
Akamai Technologies
1400 Fashion Island Blvd
San Mateo, CA 94404

Jay Parikh
Akamai Technologies
1400 Fashion Island Blvd
San Mateo, CA 94404

William E. Weihl
Akamai Technologies
1400 Fashion Island Blvd
San Mateo, CA 94404

ABSTRACT

Content delivery networks have evolved beyond traditional distributed caching. With services such as Akamai's EdgeComputing it is now possible to deploy and run enterprise business Web applications on a globally distributed computing platform, to provide subsecond response time to end users anywhere in the world. Additionally, this distributed application platform provides high levels of fault-tolerance and scalability on-demand to meet virtually any need. Application resources can be provisioned dynamically in seconds to respond automatically to changes in load on a given application.

In some cases, an application can be deployed completely on the global platform without any central enterprise infrastructure. Other applications can require centralizing core business logic and transactional databases at the enterprise data center while the presentation layer and some business logic and database functionality move onto the edge platform.

Implementing a distributed application service on the Internet's edge requires overcoming numerous challenges, including sandboxing for security, distributed load-balancing and resource management, accounting and billing, deployment, testing, debugging, and monitoring. Our current implementation of Akamai EdgeComputing supports application programming platforms such as Java 2 Enterprise Edition (J2EE) and Microsoft's .NET Framework, in large part because they make it easier to address some of these challenges. In the near future we will also support environments for other application languages such as C, PHP, and Perl.

Categories and Subject Descriptors

C.2.4 [Computer-Communication Networks]: Distributed Systems – *distributed applications, distributed databases, Web services*; D.2.11 [Software Engineering]: Software Architectures – *Java, languages, .NET, patterns*; D.2.5 [Software Engineering]: Testing and Debugging – *debugging aids, diagnostics, distributed debugging*.

General Terms

Management, Performance, Design, Reliability, Security,

Standardization, Languages

Keywords

Edge computing, grid computing, distributed applications, splitter applications, web applications, N-tier applications, utility computing, Internet applications, Web services

1. Introduction

The Web has evolved dramatically from its beginnings as a global publication mechanism. Businesses and other organizations are increasingly using the Web for online business processes, ranging from supply chain management to product configurators to customer and partner portals. The key technological shift underlying this is an increasing reliance on interactive Web applications in a strategy to grow business and/or increase operational efficiency by reaching more customers and partners. These applications serve many of the same functions as older “green-screen” applications or client-server applications, but users now interact with them via the Web.

The Internet, and by extension, the Web, is a far from optimal delivery vehicle for interactive applications. Businesses use the Internet because it is better than doing nothing: they can reach more users more cheaply than with alternatives such as private networks, dial-up, phone, or fax.

But many applications delivered over the Web still suffer from performance, reliability, and scalability problems. These problems are not new; they motivated the first wave of Content Delivery Networks (CDNs) in the late 1990's. However, the problems may be more severe now than in the early days of the Web, because the business processes implemented by many of the Web-facing applications in use today are often more mission-critical to businesses.

CDNs solved the problems of the Web for static content by delivering content from caches at the edge of the Internet, close to end users. The large scale and distributed nature of CDNs permits greater reliability. Also, delivering content from edge servers ensures faster download times, particularly if those servers are carefully chosen based on network conditions and other factors. Further, the ability to allocate resources dynamically and automatically permits enormous scalability in response to marketing successes, news events, and other events leading to “flash crowds”.

As web sites have moved to greater use of interactive applications, CDNs have continued to evolve. Caching static

content provides little help for an interactive application. The embedded images and other objects may be delivered from an edge cache, but if the base HTML page is generated by an application, user requests must still travel a long distance on the Internet to contact the origin application that generates the content. Congestion and outages on networks cause some users to see download times for a page in the 10's to 100's of seconds – enough to make a site virtually unusable for those users. Additionally, capacity limitations on the enterprise data center infrastructure limit the number of users that an application can support; for consumer-facing applications the load is particularly hard to predict, resulting in money spent on excess capacity that remains idle much of the time but still insufficient to support peak loads.

We have developed a distributed application service, *Akamai EdgeComputing*, to address these problems. EdgeComputing brings to applications the same advantages that CDNs provide for static content. Akamai's EdgeComputing distributed application service is a form of utility or grid computing (see the Global Grid Forum [3] and the Globus alliance [4] for information on many current grid computing efforts). Unlike most grid computing systems, however, it is focused on interactive business applications. It is also globally distributed to ensure that applications run close to their end users, automatically providing capacity both when *and where* it is needed.

Using EdgeComputing, parts of a Web application – and in some cases the entire application – can be distributed across the Akamai network. Many client requests can be processed completely at the edge, avoiding wide area network communication altogether. For requests that require communication with the enterprise data center (e.g., to talk to a transactional database, or to interact with legacy systems), only the raw data needs to be exchanged, not the entire HTML page, reducing the amount of data sent long-haul by one to two orders of magnitude. In addition, the same network of servers can be used to optimize the path taken over the Internet between the edge and the enterprise, avoiding congestion and outages in real-time and further reducing response time.

At its core, our EdgeComputing service requires the ability to virtualize server resources. There are many ways to do this, including virtual machine monitors (e.g., VMware [10], Xen [11]), user-mode Linux [9], and binary code rewriting (e.g., Etch [8], ATOM/OM [11]). For performance and manageability reasons, we chose not to provide each customer with a set of virtual machines for its applications. Instead, we run each customer's applications in separate application server processes, relying on kernel and operating system functionality to isolate customers from each other and to prevent runaway use of server resources. We then provide a customer application console through which a customer can view a summary of all of his running application instances, and when necessary drill down to a detailed look at individual instances. Initially, we have implemented EdgeComputing for J2EE and .NET, which provide standard programming environments that simplify the deployment and management of clustered Web applications. Our current implementation supports Apache's Tomcat server and IBM's WebSphere Application Server; support for Microsoft's .NET framework will be available later in 2004, followed by support for "native" application language environments (C, Perl, etc.).

Our original motivation for a distributed application service was to solve the performance, reliability, and scalability problems plaguing Web applications. As we discuss below, EdgeComputing has enabled our customers to ensure truly interactive sub-second response time to all users, with far greater reliability and scalability than they could cost-effectively achieve on their own. In addition, EdgeComputing has resulted in unexpected benefits, though viewed in retrospect the benefits are not so surprising. "Time to market" has been greatly reduced for many applications because customers no longer have to deal with capacity planning, acquiring, and provisioning infrastructure, thus eliminating significant risk, cost, headaches, and delays. If an enterprise still maintains a central infrastructure, they need less. Customers can focus on what they want the application to do and on building it. Deployment then involves simply publishing the application to Akamai, which manages the application infrastructure within an on-demand environment. Customers also can worry less about optimizing applications for scale, although they (and we) still need to worry about performance issues overall.

2. The EdgeComputing Model

Our EdgeComputing distributed application service is not a new programming model or a new set of programming APIs; it is a new deployment model for Web applications. Today, most Web applications are deployed within a single data center, perhaps in a cluster of machines connected by a high-speed LAN. To use EdgeComputing, a site developer typically must split the application into two components: an *edge* component and an *origin* component. The code in the edge component is deployed onto Akamai's network of servers distributed around the world (more on how this is done below); the origin part is deployed in the traditional manner within the central data center.

Some applications naturally and easily divide into edge and origin components; others require some redesign. We discuss this issue in more detail later in the paper.

We have implemented a replication subsystem for session state, permitting an application's edge components to maintain per-user state that remains available even when users are mapped to different servers; more detail on session management is in Section 6. Edge components can also use our NetStorage facilities (a distributed collection of storage silos that provides automatic replication of data) to store information submitted by users for later processing by the origin components.

Our management interfaces support staged deployment for applications. Typically, a site developer tests an application internally on test machines before deploying it to Akamai. After the application passes internal testing, the developer deploys it across the Akamai network in stages, watching for problems at each stage. More specifically, deployment involves several steps. (For concreteness, we focus here on J2EE applications.)

1. A site developer packages the edge part as a WAR (Web ARchive) file.
2. The developer publishes the WAR file to Akamai through the Akamai portal (via a Web page or Web service).
3. The Akamai portal "scrubs" the WAR file to ensure that it can run on our edge application servers. Akamai validates the WAR file and makes sure the components

are compiled for the correct application server platform.

4. The developer uses the Akamai portal to control the deployment of the application, first deploying it to a test network of Akamai machines that are identical to our production machines except that they do not serve live traffic, then to a few machines serving live traffic, then to a small but significant percentage of the network, and finally to the entire network. At each stage of the process, the developer can drive test load against the application and watch for errors, alerts, and other indications that there are problems with the application.
5. If problems are found, the developer can abort the deployment and revert back to the older application.

Deployment occurs in stages such that multiple versions of an application may be live simultaneously, a process that limits the impact of errors to as few users as possible.

Applications belonging to different customers may run simultaneously on the same machine. We have implemented a security “sandbox” to ensure that applications cannot interfere with one other or with the underlying operation of our servers. Security sandboxes are discussed in Section 4.1.

Server resources on the Akamai network are managed automatically. When the load on an application increases, additional instances of the application are started on additional servers near the users making the requests, and requests are directed to those servers. When the load drops, application instances may be stopped automatically. This automatic resource management and load balancing system is discussed in Section 4.2.

Logically, a site developer can view the Akamai system as a large virtual cluster, distributed around the world. Server resources can be added or removed from the distributed cluster at any time, dynamically and automatically. To enable debugging in such a system, our portal provides a comprehensive view of all running instances of an application. Debugging is discussed in more detail in Section 4.3.

Finally, billing for our EdgeComputing service is based on usage, measured in requests per month. Applications are allocated a certain amount of bandwidth, memory, CPU, and other resources. Applications can be given more than the standard allocation, typically for an additional application resource fee. An alternative would be to allow unlimited resource usage, and to bill for what is used. However, most customers have indicated a preference for a model in which usage per request is limited in order to avoid receiving an enormous bill at the end of the month because of an infinite loop or some other bug that creates runaway usage.

3. Example Edge Applications

We launched EdgeComputing as a service in early 2003. As of this writing, we have a large number of customers running a wide range of applications on the service. Those applications fall into several categories, summarized in the following subsections.

3.1 Content Aggregation

The simplest category of applications involves applications that have no internal databases, but simply aggregate and format

content from other sources. This category includes portals, which aggregate news, search, and other data sources to present a unified interface. This content can be transformed depending on the end user device. These applications are simple to run on the edge; data can be retrieved from the various sources using HTTP or Web services, formatted on the edge, and then returned to the user. If the data from the various sources is cacheable, most user requests can be processed completely on the edge without any long-haul communication.

Note that some portals today permit customization based on a user profile kept in a database. Today, that database would be maintained in our customer’s data center. User profiles can be retrieved from the database over the Internet (typically through a proxy servlet or similar mechanism running at the origin in front of the database), and then used to customize the page returned to the user. With some care, user profiles can also be cached on the edge, allowing most communication with the origin to be avoided altogether.

Another application in this category found on many sites is a store locator application, which allows a user to enter a location, for example, by city or zip code, and find out what stores are near that location. In many cases, the real work of this application is done by a Web service such as MapPoint [7]; the application on the customer site simply formats the results returned by the service. This is really a special case of portals, requiring only a single data source.

3.2 Static Databases

Applications with static databases can often be run entirely at the edge. Applications today are limited to databases of moderate size (100’s of megabytes). Examples include:

- Store locator – the part often done by MapPoint, where the database involves the actual stores and their locations as well as the geographical information needed to determine proximity.
- Product catalogs – except for real-time inventory information, the information in most product catalogs is relatively static and can easily be cached on the edge servers. For J2EE applications, we are working with IBM to deploy Cloudscape [5], a pure Java relational database, to enable these kinds of databases to be run alongside the edge application, accessed using JDBC. Note, that the Cloudscape edge database technology is applicable to many other types of edge applications, not just product catalogs.
- Site search – the index for searching a site can typically be cached on the edge.
- Product configurators – again, the database containing product information and the business rules controlling allowable configurations and associated pricing can often be cached on the edge.

Configurators are a little different from the other applications in this category, since they often require client session state to keep track of the configuration selections already made by the user. We replicate session state across multiple servers in order to handle when a user gets mapped to different machines over time without losing the user’s session.

3.3 Data Collection

Some applications, such as registration (for products, college courses, etc.), online applications (where one applies for something, such as college or a credit card, not in the sense of computer “applications”), and customer service (submitting a question via a web form) involve collecting data from forms filled in by users. If the application is a single-page form, the data can be collected on the edge and sent via HTTP POST (or a Web service call) directly to the origin, or the POST data could be sent to our NetStorage system. The NetStorage system replicates the data in files on disk for later retrieval by the customer.

If the application form involves multiple pages, the information entered progressively can be kept in session state on the edge. The choice of which form to present next – a choice perhaps based on the data entered so far – can be made on the edge, and the final data can again be stored on NetStorage or sent to the origin. Data validation can also be done at each step to filter out nonsensical data.

As another option, if the data is only needed for later offline “batch” processing, the edge application server can log some data for each request. This logged data is collected from all servers. Later, the collated log files are made available for processing by the origin.

Some news and portal sites, among others, use voting or polling applications to allow their users to express their opinion or preferences on various issues. This data can be collected and buffered for short periods on the edge and then sent to the origin. This allows the origin to service a small number of medium-to-large requests instead of a very large number of small requests. This reduction in turn enables the application to scale to larger loads without significant dedicated central infrastructure. The origin can then publish the aggregated data to the edge, where the application can refresh its view every few seconds to give users virtually real-time polling results. Loads on this kind of application can be particularly hard to predict and thus prohibitively expensive to provision dedicated infrastructure for peak loads.

3.4 Two-Way Data Exchange

A slightly more complex version of the data collection application involves data flowing in both directions with the behavior of the application depending on the aggregated data published in real-time by the origin. One example is online ad serving in which the ad to be served depends on the available ad inventory and which ads have been served so far. Another example is contests, in which the probability of winning may depend on the number of people who have already won. As with the voting or polling application, buffering and aggregating data in the network for brief periods can reduce the central infrastructure required by several orders of magnitude, enabling a site to handle much higher peak loads at lower expense. In the case of one customer, a contest application was designed and developed to run on Akamai EdgeComputing; it used virtually no central infrastructure, but still supported over 70 million requests during the 5-hour contest. Without EdgeComputing, a great deal of time would have been required to procure and provision the infrastructure, and it is likely that the load would have exceeded the capacity in any event, since the actual load exceeded the expected load by roughly a factor of 7.

3.5 Complex Applications

The final category is the catchall of more complex applications, including full e-commerce engines, supply chain management, customer relationship management, online banking, etc. Many of these applications have pieces that fall into the other categories above (e.g., e-commerce engines include product search, store locators, and other pieces that can be moved to the edge), but they also include pieces that rely on transactional databases. In such cases, the application must be split, typically to put the presentation layer on the edge and leave the business logic and data access layers at the origin. Even though some central infrastructure is still required, there can still be significant advantages in performance because only raw data needs to be retrieved from the origin, rather than full HTML pages. More information on best practices for how to handle these types of applications is provided in Section 5.

4. Technical Challenges

In this section we discuss a number of technical challenges faced in implementing the EdgeComputing distributed application service, including security, debugging, accounting and billing, load balancing, resource management, and monitoring, and session state replication.

4.1 Security

Customers’ EdgeComputing applications run on machines within the Akamai network, and multiple applications from different customers may run simultaneously on a machine. For this reason, each EdgeComputing customer application is executed within a security sandbox that prevents EdgeComputing applications from accessing unauthorized server resources and data, and also keeps each application from over-utilizing granted server resources. Our primary concern in isolating applications from each other is to protect against buggy code, not against malicious users; we have a business relationship with our customers that makes the deployment of malicious code unlikely.

The security sandbox performs access control checks each time an application tries to access a resource and checks limits on resource usage. For example, if an EdgeComputing application is allowed access to an area on disk, the sandbox prevents the application from reading or writing data outside the allotted disk area, limits the rate of disk access, and enforces a limit on the total amount of data stored on disk. The sandbox also limits the hosts to which the application can communicate, and it limits the number of network connections the application can have open at any given time.

To provide access control and resource management, the security sandbox makes use of facilities from the operating system such as file system quotas, user/group id permissions, and custom sandbox modules. In addition to these facilities, the sandbox can be extended with finer granularity of access control and sandboxing by making use of security capabilities provided by the application server, such as the J2EE Security Manager and policies.

To achieve better isolation, each customer’s EdgeComputing application is run in a separate process on each machine. This isolates any ill effects from a misbehaving application and prevents that misbehaving application from hampering the performance of other applications running on the system. Before an EdgeComputing application is launched, libraries and

configuration files needed by the process are stored in an area of disk allocated to the process, and permissions and quotas are set up as necessary. Also, the application's process is configured such that it cannot access files or directories outside of the allocated area on disk.

Once a process is launched for an application, the custom sandbox modules track system calls made by the server process. The sandbox modules are used to throttle access to resources and to report resource usage information to an external monitoring process that monitors all EdgeComputing applications running on the machine. The monitor process may restart, throttle, or terminate a process that tries to access forbidden resources or over-utilizes resources on the machine. It also monitors the time used by the process for each user request, and it may terminate or restart the process if requests take longer than a configurable threshold.

Since edge applications can also use session data (managed and replicated by the Akamai EdgeComputing systems), the sandbox also enforces limits on session object size to prevent over-consumption of disk and memory for storing and replicating session data. Further, the security sandbox ensures that application session objects are accessible only from the appropriate customer application by encrypting the session data using separate customer specific keys.

4.2 Load Balancing, Resource Management, and Monitoring

The problem of preventing overload of any particular server or application can be challenging enough within a single data center. In a globally distributed system, the problem is magnified. The Akamai system automatically monitors client traffic, network conditions, and application performance and automatically distributes traffic and applications as needed to prevent overload of any particular server or application instance.

Logically, the Akamai network can be viewed as a single virtual multi-computer that can run multiple EdgeComputing processes. At a more detailed level, the Akamai network is made up of many individual Akamai edge servers. Each edge server is capable of starting, stopping, and monitoring the execution of an EdgeComputing process.

Load balancing is done hierarchically, first among edge server groups (ESGs), and then within ESGs. Within an ESG, load balancing focuses on ensuring that each application is running on enough servers in the ESG to handle the load for that application directed at the ESG. Among ESGs, load balancing focuses on distributing load for different applications to ensure that no ESG runs out of capacity, at the same time optimizing for network conditions and end user performance.

At the top-level, load is dynamically mapped among ESGs using DNS, much as in the original Akamai CDN. Within an ESG, load is redistributed both using DNS (by returning more or fewer IP addresses in response to DNS requests) and by tunneling requests, as needed, from one server to another to ensure a relatively even distribution of load among instances of an application.

Within an ESG, the load-balancing algorithm uses detailed information about resource consumption by each EC application instance. Within the ESG, load-balancing agents aggregate load and capacity data and report the summarized data to the top-level

load-balancing agents. In addition, the agents analyze the local ESG data to determine how many instances of each application should be running in the ESG and whether load should be shifted within the ESG.

As mentioned previously, each edge server runs a monitor process that monitors all EdgeComputing applications running on the server. The monitor process gathers resource usage, health and event information from the OS as well as from Akamai components running in-process with each EdgeComputing application. This data is fed to the load-balancing system as well as to the Akamai reporting system. The reporting system aggregates data from all machines in the network, providing a real-time monitoring data view.

The CPU, disk, and memory usage of each application is monitored and reported. Periodic checks are made to each EdgeComputing process to make sure that it is functioning properly. Some EdgeComputing processes can be deployed to provide a periodic "heartbeat" message to be sent to the monitor process as a proactive indication that it is alive. Events generated by the EdgeComputing process (perhaps in response to a message from the Akamai network) are also monitored and reported.

For J2EE applications, a finer granularity of resource monitoring is possible. Because these application servers use Java blocking I/O APIs and a thread-per-request model, we can track and report the CPU used for each request. In addition, the memory free and total memory used inside the JVM is also monitored and reported.

All of the aggregated monitoring data is reported from each EdgeComputing process running across the entire Akamai network. This data can trigger sophisticated alerts for the Akamai NOCC. The data related to the customer's application is also presented to the customer through an application console. Many types of alerts can be generated; examples include memory utilization inside a customer's JVM, EdgeComputing process restarts because of resource over-utilization, failed customer application installs, and failed application delivery. Our customer application console also contains real-time information for each running instance of the customer's applications, showing the current state of the process, memory and CPU usage, and the number of client requests served by the process.

4.3 Debugging

Each EdgeComputing process can write logs to disk that can then be accessed through the customer application console/portal. This allows the EdgeComputing application developer to write diagnostic information such as stack traces and event messages to a log file that can be viewed through the Akamai portal. Customers can see data for each instance of their applications running on the Akamai network. They can drill down on a specific instance on a specific machine and retrieve log files from that machine.

Many JVMs allow programmatic generation of diagnostic information about the JVM and the application running inside it. This information includes thread dumps and JVM heap dumps. Our EdgeComputing implementation for J2EE allows this information to be requested through the customer application console for a specific application instance; the JVM writes the information to a file that can then be accessed by the customer in

the same manner as standard debug log files. A customer could retrieve this information from an application instance to diagnose a problem flagged by alerts or by other information available through the portal.

Akamai provides a staging test network that replicates the production environment, except that it does not service live load. This permits functional and load testing of applications to be done in a safe environment that is as close as possible to the production environment.

4.4 Application Session State

End user session state is a common aspect of Web applications developed using J2EE or .NET (and others). This application state is conventionally stored in-memory in the application server process that is handling the end user request. By default, most application servers will not replicate or persistently store this session data, so it is not made available to other application instances in a clustered environment. This introduces a single point of failure for users mapped to a particular application instance, since if the server goes down, the client session data would be lost. In clustered enterprise environments, one can make use of in-memory replication sub-systems or a database to replicate and store the created session state. This can provide a higher level of fault tolerance to the application.

Akamai EdgeComputing dynamically maps end users to the appropriate application instance across the entire network. This introduces an interesting problem of session affinity, since we can move end-user traffic from server to server. Having one centralized database to provide persistent session storage would add unnecessary latency in servicing the request and might even render worthless the benefits of moving an application to the edge.

EdgeComputing provides an edge session replication system that allows for client session objects to be replicated across servers in an ESG as well as across ESGs. This system provides the application server session interface with an in-memory cache to reduce latency. The locally cached session object is checked with the system replicated session object to validate that the session object is “current”, and if so, the object can be used by the EdgeComputing application (thus, avoiding the deserialization of the object data). If the end user is mapped to another server, the application server on that server checks its local cache.

Remapping occurs when machines suffer hardware or software failures, when the network between a user and a machine becomes congested, and when load needs to be redistributed to prevent machines from becoming overloaded. However, remapping is relatively infrequent, so it is likely that consecutive requests in a user’s session will contact the same server. In the common case, the user’s session object will be found in the local cache. The first time the user has been mapped to this machine, the application server must retrieve the user’s session object from the session replication system. If the application modifies the session object, the updated object is sent asynchronously to the edge session replication system.

5. Edge Application Best Practices

The EdgeComputing development model remains the same as for centralized applications; it does not require the use of any proprietary APIs. The deployment model changes, not the

programming model. If an existing application follows well-known component programming best practices, adapting it for EdgeComputing is easier. For example, EdgeComputing for J2EE enables the execution of J2EE Web tier application components — JSPs, servlets, tag libraries, and JavaBeans.

In order to provide more context, this section focuses on describing the best practices specifically related to development for EdgeComputing for J2EE using Tomcat or IBM’s WebSphere Application Server. These best practices have been established working jointly with IBM and described in detail in [2]. In general, EdgeComputing applications are architected as two cooperating sub-applications: an edge-side application running on Akamai and a corresponding origin-side application.

5.1 Presentation Components on the Edge

The most common EdgeComputing model is to deploy the presentation components of an application onto the Akamai edge servers and to cache access to origin data via the Java Web services client model. Typically, the Web application will be developed using a framework based on the Model-View-Controller (MVC) architecture.

Jakarta’s Struts framework [6], a common open-source framework for building Web applications based on the MVC pattern, is well suited for EdgeComputing. Struts provides its own Controller component via a servlet. For the View, Struts-based applications often leverage JSPs to create the end user response. The Model component is commonly represented by JavaBeans. These Model JavaBeans may be self contained or represent façades for other components like JDBC or EJB.

The View and Controller components of a Struts application are good candidates for distributing on EdgeComputing. These components run on the Akamai edge servers and can interact with Model components running at the origin (EJBs). Depending on the functionality of your application, the extent to which these applications can move onto EdgeComputing will vary. The edge View and Controller components are bundled, along with other Java classes, into a Web application archive (WAR) and deployed onto the EdgeComputing network.

With EdgeComputing today, EJBs commonly remain running at the origin and are made accessible to the edge application via Web services interfaces. A session bean façade can be used to expose the necessary business logic as a Web service to the edge application. The edge application makes a Web service call back to the origin to invoke the appropriate logic through a session bean façade, perhaps made visible through a servlet running on the origin application servers. An edge application can and should cache the results of SOAP/HTTP(S) requests to minimize interactions with the origin.

In addition to Web services as a communication channel, our EdgeComputing distributed application service supports other standard communication protocols, including the following:

- **HTTP(S)** – An edge application can make HTTP(S) requests to the origin to request content or data objects. HTTP responses from the origin should be cached on the edge when possible, so content or data objects can be persisted on the edge across user requests to further reduce the load on the origin. A developer could leverage Akamai’s content caching system for content objects (HTML fragments). Akamai provides an

invalidation control mechanism (via a Web service) to customers, so that customer can control the invalidation of any content objects cached on the Akamai edge servers. Additionally, an edge application can leverage caching Java objects in the edge JVM instance.

- **JDBC** – Akamai provides a JDBC driver that allows edge applications to tunnel JDBC queries to the origin via HTTP(S). If an application has already been developed using JDBC for data transaction with a database, the JDBC/HTTP mechanism will make it easier to adapt an application for the edge. Further, JDBC query responses can be configured to be cached on the Akamai edge servers similar to HTTP responses.
- **RMI** – Edge application components can use RMI tunneled over HTTP(S) to communicate with the origin component. In this configuration, a servlet runs at the origin, intercepts the RMI request from the edge, and translates the request into the appropriate method calls to the business tier.

Rather than using RMI, JDBC, and other protocols in their native mode to communicate from the edge to the origin, we tunnel them over HTTP(S) to avoid problems with firewall configurations and to avoid requiring that enterprises make their databases and legacy systems directly accessible on the Internet.

5.2 Data Access on the Edge

Any of the HTTP-based protocols above are useful interfaces that allow you to “bridge” your applications from the edge to the origin, but it is still important to avoid excessive communication for edge-origin requests, as otherwise end-user latency and origin utilization will increase. Because there is an absolute cost for every roundtrip from the edge to the origin, calls should be as “coarse-grained” as possible. Multiple requests to the origin that are required to service a single end-user request should be bundled, if possible, and edge caching should be used to store data and other objects across requests.

The Akamai EdgeComputing platform provides caching mechanisms to persist application data to minimize the interaction and load on the origin infrastructure on a request-by-request basis.

- **Client Session** – EdgeComputing supports replicated client sessions. Session objects are most commonly used to keep per-user state information. Akamai enforces a size limit to the actual serialized object data for performance reasons.
- **Client Cookies** – An edge application can store some user specific data in user cookies or URL tokens. Privacy and security concerns, however, may prohibit an enterprise from using this mechanism.
- **Object Caching** – As previously mentioned, an edge application can make requests for data or content (using HTTP, Web Services, or JDBC) and these objects can be stored in the edge node’s cache.

Another powerful edge data capability employs IBM’s 100% pure Java Cloudscape DBMS to host infrequently changing, read-only data in an edge database. In this model, application data is exported as part of the application WAR file. By installing Cloudscape on the edge, even the Model components of an MVC

application can be run on EdgeComputing. An edge application can make use of JavaBeans and JDBC as Model components with Cloudscape as the DBMS to further reduce the communication to the backend enterprise systems. One current limitation of this model is that any time the application data changes, the application must be redeployed to the EdgeComputing network.

6. Future Work

At its core, EdgeComputing involves virtualizing system resources and allocating them dynamically to different applications. Simple virtualization, however, is not enough; for more than a few customers to be willing to use EdgeComputing, it must be easy to manage applications running on it. Our goal has been to make it as easy to manage an application running on EdgeComputing as it is to manage an application running on a cluster of machines in a central data center. This includes support for testing, monitoring, and debugging, as well as support for splitting an application into “edge” and “origin” pieces and orchestrating communication between the edge and the origin.

We are currently working on a number of areas to simplify the management of distributed applications on our EdgeComputing service, including:

- Further simplifying management of multiple versions of applications.
- Providing deeper insight into the performance of applications on our edge servers to make it easier for customers to tune the performance of their applications.
- Handling multiple versions of application servers.
- Supporting the full J2EE specification, including EJBs and JMS.
- Synchronizing session state between the edge component and the origin component of an application.
- Making data caching and replication on the edge transparent for more patterns of data access.
- Providing pre-integrated access to key services required by business applications (e.g., identity management, access control, and payment processing).

7. Summary and Conclusions

Akamai’s EdgeComputing service allows enterprises to deploy and run Web applications on a globally distributed computing platform. Running applications on servers at the edge of the Internet in our distributed application service provides many advantages for an enterprise including capacity on demand, better end user performance, higher application availability, and faster time-to-market.

Development of this distributed application platform presented several 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.

There are several types of applications that can benefit from this distributed application platform. By following some basic edge application best practices, such as minimizing communication with the origin and caching data on the edge, many important business applications can leverage Akamai's EdgeComputing distributed application service.

8. ACKNOWLEDGMENTS

Akamai's EdgeComputing service would not exist without the incredible efforts of the many people who have contributed to it, including Andy Berkheimer, Annie Boyer, James Chalfant, Eddy Chan, Andy Ellis, Liz Greene, Dan Hang, Marty Kagan, Nate Kushman, Erik Nygren, Nicole Peill, Vasanth Pichai, M. C. Ramesh, Eddie Ruvinsky, Sudesh Saoji, Alex Sherman, Danner Stodolsky, Ashis Tarafdar, and Kieran Taylor. Our partners at IBM have also contributed enormously to the service by helping integrate IBM's WebSphere Application Server into our EdgeComputing service and by working with us on numerous sample applications and customer engagements. In addition, we are grateful to Gerry Cuomo, Michael Jacob, and Raymie Stata for helpful comments on this paper. Finally, we must acknowledge the early contributions of the late Danny Lewin, who believed from the beginning that EdgeComputing would be valuable to our customers.

9. REFERENCES

- [1] Barham, P., Dragovic, B., Fraser, K., Hand, S., Harris, T., Ho, A., Neugebauer, R., Pratt, I., and Warfield, A. *Xen and the Art of Virtualization*. ACM Symposium on Operating Systems Principles, Bolton Landing, NY, October 19-22, 2003.
- [2] Cuomo, G., Martin, B., Smith, K., Ims, S., Rehn, H., Haberkorn, M., and Parikh, J. *WebSphere Capacity - On Demand: Developing EdgeComputing Applications*. October 2003. (http://www7b.software.ibm.com/wsdd/library/techarticles/0310_haberkorn/haberkorn.html)
- [3] Global Grid Forum. <http://www.gridforum.org>.
- [4] The Globus Alliance. <http://www.globus.org>.
- [5] IBM Cloudscape. <http://www-306.ibm.com/software/data/cloudscape/>.
- [6] Jakarta Struts. <http://jakarta.apache.org/struts/>.
- [7] MapPoint service. <http://mappoint.msn.com>.
- [8] Romer, T., Voelker, G., Lee, D., Wolman, A., Wong, W., Levy, H., and Bershad, B. *Instrumentation and Optimization of Win32/Intel Executables Using Etch*. 1997 Usenix NT Conference.
- [9] User-Mode Linux. <http://www.usermodlinux.org>.
- [10] VMware. <http://www.vmware.com>.
- [11] Wall, D. W. Systems for late code modification. In Robert Giegerich and Susan L. Graham, eds, *Code Generation - Concepts, Tools, Techniques*, pp. 275-293, Springer-Verlag, 1992. Also available as HP/Compaq WRL Research Report 92/3, May 1992.