A Developer’s Guide to On-Demand Distributed Computing

Best Practices for Design and Deployment of High-Performance J2EE Applications
Table of Contents

Introduction.................................................................................................................................2

J2EE Internet Application Performance..................................................................................3

On-Demand Distributed Computing (ODDC)............................................................................4

ODDC Application Development...............................................................................................4

Presentation Components on the Edge......................................................................................6

Data Access on the Edge.............................................................................................................7

EdgeComputing Application Examples.....................................................................................8

EdgeComputing Application Deployment................................................................................9

Summary......................................................................................................................................11
Introduction

Before applications can move from pilot and into production, developers and architects face the inevitable and time-consuming question of what infrastructure and underlying capacity will be needed to support their apps. This capacity-planning picture is not getting any prettier either. As all business applications are increasingly Internet-enabled and now available to a global user base, this becomes more of a "guestimation" than exact science.

On-Demand Computing offers the promise of providing computing cycles on a pay per use basis like electric or gas utilities. The ability to scale "On Demand" is great but it doesn't guarantee performance. This is especially true for Internet applications. Because utility computing doesn't necessarily place resources close to requesting users, it still forces all requests to a central point for processing. To successfully support Web-based applications and avoid the Internet bottlenecks inherent with "silo" serving, a distributed computing model is required.

Using On-Demand Distributed Computing, applications not only scale on demand -- they also avoid the inherent bottlenecks on the Internet. With a few adjustments in application development and design, developers and architects can propel applications into production without spinning cycles on costly infrastructure decisions. Most importantly, the On-Demand Distributed Computing model boosts performance so that applications are never "dumbed-down" to handle the vagaries of the Internet. Developers gain the freedom to create innovative applications in far less time than possible with traditional solutions.

This paper outlines the best practices for Java coding so that applications are easily deployed in the On-Demand Distributed Computing model. These practices involve usage of the existing set of services available in Java 2 Enterprise Edition (J2EE) application server containers. In many instances applications are already viable for On-Demand Distributed Computing. In others, following a few J2EE best practices have them ready for deployment in short order.
**J2EE Internet Application Performance**

A quick Internet search on “J2EE performance” proves that there is no shortage of advice to developers and architects looking to get more from their application server. Nearly all of the Internet pundits and vendors focus on server-side solutions to the problem. These typically involve improved coding practices (exception handling, garbage collection, etc.) or improved use of the existing methodologies available in J2EE (proper use of server containers). These solutions can indeed improve performance in the confines of a LAN but it is a different story when applications are exposed to the Internet through Web Services or other means. In the case of J2EE Internet applications, coupling J2EE best practices with a distributed computing platform can produce dramatic improvements in performance and reliability while actually lowering overall infrastructure costs. Here’s why:

The infrastructure to deliver J2EE Web applications typically includes a wide range of technologies including load balancers, HTTP Web servers, caching servers, messaging systems, transaction-processing monitors, application servers, and databases. A typical enterprise application infrastructure is shown in Figure 1.

**Figure 1 – Typical J2EE Internet Infrastructure**

As performance and geographic reach requirements expand, Internet application infrastructure becomes increasingly heterogeneous and difficult to scale. IT managers continually must evaluate capacity plans to keep pace with the expected peak demand, and the number of “moving parts” increases points of failure.

Pre-provisioning extra capacity as insurance against overload is financially unacceptable for most enterprises. In addition, it prevents innovation, as every new application requires a business case to prove its worth. Ideally, enterprises want computing resources when – and only when - they are needed; they do not want to buy extra infrastructure that sits idle when not needed. As seen in Table 1, below, a recent study conducted by IBM, demonstrates that Intel and Unix servers deliver sub-10% utilization rates.
“On-demand” computing provides better utilization of computing resources and represents a model in which computing resources are brought into service as needed.

**On-Demand Distributed Computing (ODDC)**

Rather than focus on server-side technologies that tend to deliver marginal performance fixes, architects can design applications such that they leverage the benefits of a distributed computing platform that extends beyond the server.

In general, there are two ways in which On-Demand Computing is being made available to enterprises: through centralized and distributed architectures. Centralized approaches to utility computing are ideal for applications that reside on a LAN or in instances where user distribution is very low. With the burgeoning number of Web Services and Internet-enabled applications, this isn’t always the case as users are typically spread across the Internet.

In contrast to centralize approaches, On-Demand Distributed Computing places processing close to users so that application performance and reliability are improved. Developers that factor this model into design stages are able to launch applications without the normal delays of infrastructure capacity planning. Internet-based applications move from pilot to production more quickly and inherently have higher performance and reliability.

**ODDC Application Development**

Akamai’s EdgeComputing service is a leading example of an On-Demand Distributed Computing platform. Deployed at the “edge” of the network – close to users’ access points - it EdgeComputing consists of tens of thousands of servers in over 1,100

### Table 1: Typical Infrastructure Utilization

| Source: IBM Scorpion White paper: Simplifying the IT Infrastructure, 2002 |
|---|---|---|
| **Mainframes** | 5-100% | 70% | 60% |
| **Unix** | 50-70% | 10-15% | <10% |
| **Intel-based** | 30% | 5-10% | 2-5% |
| **Storage** | N/A | N/A | 52% |
networks around the world. J2EE application processing is distributed across this platform so that computing is available “on-demand” close to requesting users. These globally distributed edge servers implement industry standard protocols to support tasks as simple as page assembly or as complex as J2EE processing. For example, Edge Side Includes (ESI) is an open standard specification being co-authored by Akamai, ATG, BEA Systems, IBM, Interwoven, Oracle, and Vignette. This simple markup language defines Web page components for dynamic assembly and delivery of Web applications at the edges of the Internet.

To further extend distributed computing capabilities beyond the presentation tier, Akamai supports two versions of J2EE application servers: Tomcat and IBM WebSphere. In the latter, IBM and Akamai have integrated and deployed WebSphere Application Server (WAS), Version 5.0, onto this distributed computing platform. This service – Akamai EdgeComputing powered by WebSphere – enables enterprises to run J2EE Web tier applications in an on-demand WebSphere environment and to consume Internet computing resources on a pay-as-you-go basis.

A typical Akamai EdgeComputing environment, shown in Figure 2, consists of the end user using a browser, the enterprise (origin) running business logic, legacy systems and databases, and a distributed network of servers running an embedded WAS or Tomcat server that supports the J2EE web application programming model.

Figure 2: Akamai EdgeComputing powered by WebSphere
The EdgeComputing development model remains standard J2EE and does not require the use of any proprietary APIs; it is the deployment model that changes, not the programming model. If applications generally follow J2EE component programming best practices, adapting the existing application for EdgeComputing will be easier. EdgeComputing extends the WebSphere application-programming platform to enable the execution of J2EE web tier application components — JSPs, Servlets, Tag libraries, and JavaBeans.

Development for EdgeComputing still relies on standard J2EE development tools and best practices in developing applications, but one must architect edge-enabled applications as two cooperating sub-applications: an edge-side application running on EdgeComputing Platform and an enterprise-side application.

Presentation Components on the Edge

The most common EdgeComputing model is to deploy the presentation components of your application onto the Akamai edge servers and to cache access to enterprise 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\(^1\) framework, an 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 of class ActionServlet. This Servlet is configured by defining mappings, ActionMappings, between the requested URI to the class name of an Action class. For the View, Struts can leverage JSPs and provides a very flexible and powerful set of JSP Tag libraries that allow you to build the View part of an MVC application without embedding Java code directly into the JSP itself. The Model component is commonly represented by JavaBeans. These Model JavaBeans may be self-contained or actually 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 enterprise (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.

In a typical enterprise environment, the Web tier will use RMI to communicate with the business tier. For example, a Servlet may use RMI-IIOP (Internet Inter-ORB

\(^1\) For more information regarding Struts, please see http://jakarta.apache.org/struts//.
Protocol) to call an EJB. However, with Akamai EdgeComputing powered by WebSphere, EJBs remain running at the enterprise 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 enterprise to invoke the appropriate logic through a session bean façade, perhaps made visible through a Servlet running on the enterprise application servers. The edge application can use JAX-RPC to call the Web Service at the enterprise via SOAP/HTTP(S). Akamai EdgeComputing powered by WebSphere enables an edge application to cache the results of SOAP/HTTP(S) requests to optimize interactions with the enterprise.

In addition to Web Services as a communication channel, Akamai supports other standard communication protocols, including the following:

- **HTTP(S)** – An edge application can make HTTP(S) requests, using HTTP URLConnection class for example, to the central enterprise to request content or data objects. Any content fragments can be fetched and used to build the end user response. Data objects, such as XML files, can be fetched and then be transformed (XSLT) or parsed (JAXP) to create the end user response. HTTP responses from the enterprise can be cached on the edge, so content or data objects can be persisted on the edge across user requests to further reduce the load on the enterprise.

- **JDBC** – Akamai provides a Type 3 JDBC driver that allows edge applications to tunnel JDBC queries to the enterprise 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. The edge application can still use JDBC and benefit from the relational data form. JDBC query responses may be configured to be cached on the Akamai edge servers as well.

- **RMI** – WebSphere applications running at the edge can use RMI-IIOP tunneled over HTTP(S) to communicate back to the enterprise. In this configuration, a Servlet runs at the enterprise, intercepts the RMI requests from the edge, and translates the requests into the appropriate RMI method calls to the business tier. Since RMI calls are set to be un-cacheable in the EdgeComputing environment, it is recommended to use Web services over RMI to achieve better performance.

### Data Access on the Edge

Any of these HTTP based protocols are useful interfaces that allow developers to “bridge” applications from the edge to the enterprise, but it is still important to avoid excessive communication for edge-origin requests, as end user latency and origin utilization will increase. Since there is an absolute cost for every roundtrip from the edge to enterprise, calls should be as effective as possible. Requests should be bundled, if possible, and edge caching should be used to store data and other objects across requests.
The EdgeComputing platform provides caching mechanisms to persist application data to minimize the interaction and load on the enterprise infrastructure on a request-by-request basis.

- **ServletContext** – Any data that can be used across users for a particular application can be stored in the application’s ServletContext object.

- **HTTP Session** – Akamai EdgeComputing powered by WebSphere supports the standard HTTP Session interface. The platform provides replication for HTTP Session objects across the Akamai edge server nodes. HTTP Session objects are most commonly used to keep per user state information, but keeping them small in size (kB) is important for performance reasons.

- **HTTP Cookies** – An edge application can store some user specific data in user cookies or URL tokens. Although, privacy and security concerns may prevent the use of cookies to store data in some situations.

- **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 into a JAR file (done by the enterprise application developer), and this “database in a JAR” file is bundled into the edge WAR file under the standard WEB-INF/lib directory. The application’s data source has a fixed configuration corresponding to the location of the database within the JAR file, and the application and corresponding configuration are deployed in the WAR to the edge servers.

By using Cloudscape on the edge, even the Model components of an MVC application can be distributed 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 disadvantage of this model is that any time the application data changes, the application must be redeployed to the EdgeComputing network. Future integration development between IBM and Akamai contemplates enabling Cloudscape to function as a database query cache at the edge of the network. This will enable dynamic caching of query results depending on the SQL statements issued by the edge application.

The following summary outlines suggested best practices given the type of data access required by an application running on the edge.

**EdgeComputing Application Examples**

EdgeComputing powered by WebSphere enables a powerful new deployment model for
J2EE Web applications. The following examples describe some applications modeled to run on EdgeComputing and illustrate the use of WebSphere Web Services and Cloudscape in EdgeComputing applications.

- **Product Catalog** – a product catalog browsing application can run almost entirely on the edge in the EdgeComputing environment. Since most product catalogs consist of relatively static product data (not including inventory information), the edge application can utilize Cloudscape as the local DBMS. The data can be bundled into the edge WAR along with the catalog browsing presentation components. Using this deployment model, it is feasible for the end user browsing interaction to be handled entirely by the edge application. When a user is ready to purchase any selected items, the edge application tunnels back to the enterprise for order processing.

- **Marketing Promotional Contest** – an enterprise wants to conduct a large-scale marketing promotion to give away a certain new product. Because of the uncertainty of the number of end user contestants, an on-demand edge application is extremely beneficial to assuring a successful outcome. In this scenario, the application might have “random selection” logic to determine if an end user is a winner. An EdgeComputing application can be designed and developed to execute this logic on the edge, offloading the load from the enterprise. In addition, the corporate marketing team can implement various controls on how long the contest runs, how many products are given out, the rate at which they are disbursed, or other controls. The edge application executes the corresponding business logic entirely on the edge and retrieves the control parameters from the enterprise via Web Services calls.

**EdgeComputing Application Deployment**

Once designed, developed and tested, the edge sub-application is uploaded and provisioned on the Akamai EdgeComputing powered by WebSphere network. Akamai has developed a provisioning mechanism that is available via two interfaces. The conventional interface involves using a browser-based GUI on the Akamai enterprise portal. A second interface involves uploading and provisioning via SOAP-based Web Services that can be invoked programmatically.

Further, IBM and Akamai have developed a plug-in for WebSphere Studio Application Developer (WSAD) in the form of a GUI "export wizard". A screenshot of the wizard is shown in Figure 3. This plug-in securely invokes the EdgeComputing provisioning Web Service, and it allows developers to deploy their applications directly from WSAD environment. In addition to the upload function, the plug-in is being enhanced to provide validation and simulated unit edge test environment for the WSAD environment.
Figure 3: Screenshot of (WSAD) GUI “export wizard”
Summary

Before application architects embrace “On Demand”, they must realize that it this new deployment model does not guarantee Internet application performance. To successfully support Web-based applications and avoid the Internet bottlenecks inherent with “silos” serving, a distributed computing model is required.

Using On-Demand Distributed Computing, Internet applications can deliver new levels of performance and reliability regardless of user location or load. By using the existing set of services available in J2EE application server containers, applications are easily deployed on a distributed computing network like Akamai’s.

This proven model means that enterprises do not need to expend time or money on complex capacity planning. For developers, the On-Demand Distributed Computing model boosts performance so that applications are never “dumbed-down” to handle the vagaries of the Internet. Enterprises can create innovative applications in far less time than possible with traditional solutions.

Working with IBM, Akamai has developed a new deployment model for J2EE web applications. Akamai EdgeComputing powered by WebSphere service allows for application components to migrate to the edge. By extending the Tomcat and WebSphere programming platforms to run on the Akamai edge network, enterprises are able to attain higher levels of performance, reliability, global reach and ultimately, achieve scalability on-demand. Akamai EdgeComputing powered by WebSphere helps enterprises respond to a growing need for “on-demand” computing and will help many enterprises adapt their business functions to the edge.

About Akamai

Akamai® is the global leader in distributed computing solutions and services, making the Internet predictable, scalable, and secure for conducting profitable e-business. The Akamai on demand platform enables customers to easily extend their Web operations - with full control - anywhere, anytime, without the cost of building out infrastructure. Headquartered in Cambridge, Massachusetts, Akamai serves hundreds of today’s most successful enterprises and government agencies around the globe. Akamai is The Business Internet. For more information, visit www.akamai.com.