Tuning WebSphere Application Server Cluster with Caching

Contents

Tuning WebSphere Application Server Cluster with Caching

This paper analyzes parameter and configuration variations to identify what influences the throughput when caching is enabled in a WebSphere® Application Server 6.1 cluster running the Trade workload in a secure environment.

Published May 2009

These components are:

- SUSE Linux® Enterprise Server (SLES) 10 SP1 and Red Hat Enterprise Linux (RHEL) 4 ES
- WebSphere Application Server
- IBM® DB2® Universal Database™ (DB2®) on Linux for IBM System z®

This paper discusses various cluster tuning parameters with regards to caching, and presents the results of various test scenarios.

To view or download the PDF version of this document, click on the following link:

Tuning WebSphere Application Server Cluster with Caching (about 2.8 MB)

Introduction to tuning the WebSphere Application Server cluster with caching

The purpose of this project was to analyze parameter and configuration variations to identify what influences the throughput when caching is enabled in a WebSphere Application Server 6.1 cluster running the Trade workload in a secure environment.

Objectives

An earlier project showed that when a single WebSphere Application Server 6.0.2 uses caching, it provides a significant throughput improvement. The first experience with a WebSphere cluster showed that caching in the cluster is much more complex, where overhead can easily increase in a way that negates any advantage gained with caching.

For previous information on WebSphere Application server in a secure environment see:

download.boulder.ibm.com/ibmdl/pub/software/dw/linux390/perf/ZSW03003USEN.PDF

Summary for the WebSphere Application Server cluster with caching tests

This is a summary of results and recommendations as a result of running the Trade workload on the four node WebSphere Application Server cluster with caching enabled.

These test results and recommendations are Trade specific. Parameters useful in this environment might be useful in other environments, but because caching benefits and costs are highly dependent on application usage and system configuration, you will need to determine what works best for your environment. For detailed test results information, refer to "Test case scenarios and results" on page 27.

The following are summary results related to hardware and hardware configuration:

- The WebSphere proxy server obtained a better overall level of throughput than the WebSphere Edge Service Caching Proxy Server.
- The IBM System z10™ system obtained a significant throughput advantage over the IBM System z9® system.
- The z/VM® VSWITCH LAN configuration resulted in higher throughput than the Guest LAN feature.

The following are summary results related to software and caching:

- Any form for caching: Command caching, Command and servlet caching, or Distributed Map caching, resulted in a significant throughput improvement over the no caching case.
- Distributed Map caching resulted in the highest throughput improvement.
- A Dynamic Cache size of 10000 or greater produced the best throughput.
- The smaller Dynamic Cache sizes benefit from enabling DynaCache disk offloading. This is especially true for a transaction mix that is mostly read only.
- Configuring disk offload is not related to overhead in the tests that were run. This feature could be used to run with a significantly smaller cache size. For example, 2000 statements would save memory without causing a performance degradation.
- For this environment, the default JVM garbage collection policy resulted in the highest throughput.
- In this environment with the Trade application, the non-shared cache replication policy resulted in the highest throughput.
- The shared-pull and shared-push-pull replication policies resulted in significant throughput degradation.

Based on the results of the test scenarios, the following is recommended:

- Use some form of caching whenever possible. The additional development effort to add Distributed Map caching in the application can result in a measurable throughput improvement.
- Enable disk offloading when using small Dynamic Cache Sizes.
- Use z/VM VSWITCH instead of a guest LAN.

Benefits of using z/VM

These are the benefits of using z/VM in a Linux on an IBM eServer™ zSeries® environment.

The following list shows how environments, such as the one modeled here, could benefit from z/VM.

- Using z/VM can help increase speed for data transfers, because virtual LANs are used, which reduces the latency associated with a physical network and memory-to-memory data transfer rates could be achieved.

- Using z/VM can help reduce administration effort.
- Systems can be easily cloned.
- Isolated networks can be easily implemented without physical hardware such as:
  - Network cards
  - Switches
  - Cables
- Using z/VM can simplify administration due to the fact that there is only one physical system.
- Using the Performance Toolkit for z/VM provides a single point to easily monitor all servers.
- Using z/VM allows very efficient use of the hardware, by sharing the physical hardware between the guests.
- Using virtual CPUs allows consolidation of several under utilized systems on a small number of CPUs with higer utilization, which reduces cost and overhead.
- It is possible to use more virtual hardware than available physical hardware.

Hardware equipment and software environment

This section provides details on the hardware and software used in performing the tests.

Topics include:

- Server and client hardware used.
- Software used.
- Test environment.
- A description of the workload used.

Host hardware

The WebSphere Application Server tests were performed with a customer-like environment. This is the server hardware that was used.

One logical partition (LPAR) on a 16-way IBM System z9, type 2094-S18 equipped with:

- Eight physical CPUs, dedicated
- 12 GB memory
- 2 GB expanded memory
- One OSA-Express 2 Gb Ethernet card
- Eight FICON® Express™ channels for z/VM
- Eight FICON Express channels for the guests

One LPAR on a 16-way IBM System z10™ type 2097-E26 equipped with:

- Eight physical CPUs, dedicated
- 12 GB memory
- 2 GB expanded memory
- One OSA-Express 2 Gb Ethernet card
- Eight FICON Express channels for z/VM
- Eight FICON Express channels for the guests

Network setup

To perform the WebSphere Application Server tests, this network configuration was used.

- The z/VM LPAR was connected to the client using a Fibre Gigabit Ethernet Interface.
- The z/VM guests used two virtual guest LANs configured as type HiperSockets™, with a maximum frame size (MFS) of 24 KB. An alternative was an environment configured using the z/VM virtual network feature VSWITCH.

Storage server setup

To perform the WebSphere Application Server tests, a customer-like environment using IBM System Storage™ servers was created.

IBM System Storage DS8300 2421 Model 932 servers were used:

- IBM 3390 disk models 3 and 9
- Physical DDMS with 15,000 RPMs

Client hardware

To perform the WebSphere Application Server tests, this client hardware was used.

The environment had one x330 PC with two 1.26 GHz processors, and a 1 Gb Ethernet adapter.

Software (host and client)

This is a list of the software used to perform the WebSphere Application Server tests.

Software (host and client)

Table 1 lists the software used in the WebSphere Application Server test environment.

Table 1. Host and client software used in the WebSphere Application Server test environment

Product	Version and Level
Host	
Apache HTTP Server	2.0.49 (64–bit)
DB2	v9.5
Firewalls	iptables-1.2.5–13.2 (part of SLES 10 SP1)
SUSE Linux Enterprise Server	SLES 10 SP1
WebSphere Network Deployment for Linux • Application Server • Deployment Manager • Proxy Server • Web Server plugin	6.1.0.15 (31–bit)
WebSphere Application Server Edge Component Caching Proxy Server	6.1.0.15
z/VM	5.3

Table 1. Host and client software used in the WebSphere Application Server test environment (continued)

Product	Version and Level
Host	
Client	
Red Hat Enterprise Linux	RHEL 4 ES
WebSphere Studio Workload Simulator	iwl-0–03309L

Overview of the Caching Proxy Component

The caching proxy server is used to handle and validate Internet requests.

In an enterprise environment, a proxy server is a server that acts as an intermediary, typically placed in a demilitarized zone (DMZ). This DMZ is between the Internet and the server environment in the internal zone, providing the business services. It validates the request for an Internet service.

If the request passes filtering requirements, the proxy server forwards it to servers in the internal (secure) zone and acts as the requester. This mechanism prevents direct access from the (insecure external zone) to the sensitive servers in the internal zone (see also "Test environment" on page 7).

The proxy servers used here can also improve performance by caching content locally.

The two main advantages of using a proxy server are system security and performance:

- Security: A proxy server provides an additional layer of security and can protect HTTP servers further up the chain. It intercepts requests from the client, retrieves the requested information from the content-hosting machines, and delivers that information back to the client. If you are using a firewall between the reverse proxy server and the content HTTP server, you can configure the firewall to allow only HTTP requests from the proxy server.
- Performance: A proxy server can increase the performance of your WebSphere Application Server in several ways.
  - Encryption/SSL acceleration: You can equip the proxy server with SSL acceleration hardware that can improve the performance of SSL requests.
  - Caching: The proxy server can cache static content to provide better performance.
  - Load balancing: The proxy server can balance the workload among several content HTTP servers.

WebSphere Edge Components Caching Proxy

The WebSphere Edge Component Caching Proxy (CPS) reduces bandwidth usage and improves a Web site's speed and reliability by providing a point-of-presence node for one or more backend content servers. The CPS can cache and serve both static content and content dynamically generated by the WebSphere Application Server.

The proxy server intercepts data requests from a client, retrieves the requested information from content-hosting machines, and delivers that content back to the

client. Most commonly, the requests are for documents stored on Web server machines (also called origin servers or content hosts) and delivered using the Hypertext Transfer Protocol (HTTP). However, you can configure the proxy server to handle other protocols, such as File Transfer Protocol (FTP) and Gopher.

The proxy server stores cache-able content in a local cache before delivering it to the requester. Examples of cache-able content include static Web pages and JavaServer Pages files that contain dynamically generated, but infrequently changing, information. Caching enables the proxy server to satisfy subsequent requests for the same content by delivering it directly from the local cache, which is much quicker than retrieving it again from the content host.

There are several plugins for the Caching Proxy and for additional functionality to the proxy server, but only the default setup was used.

WebSphere Proxy Server

WebSphere Proxy Server (PS) is a new type of server supported in the WebSphere Application Server Network Deployment (ND) package (in version 6.0.2 and later). This Proxy server receives requests from clients initially on behalf of content servers and work load manages, and routes the requests across content servers depending on the policies and filter classification definitions.

WebSphere Proxy servers can secure the transport (using SSL), secure the content, and protect the identity of application servers using the response transformation feature (URL rewriting). The Proxy server can also cache responses to improve throughput and performance. Another good feature to note is SSL offload at the Proxy server. When using this feature you can terminate an SSL (HTTPS) connection at the proxy server after receiving the request from the client and use HTTP as transport protocol between proxy server and the content servers (which are application servers). You can administer and configure this Proxy server from the deployment manager's administrator console (or wsadmin) in an ND environment.

This Proxy server is much more capable than the reverse proxy servers (the Edge caching server and the WebSphere plugin) with its advanced configuration capabilities, dynamic routing policies, and integrated system management in ND topology. It is interesting to note that the Proxy server can also route requests across multiple cells and supports session affinity and failover.

Configuration and test environment

This section provides information necessary for setting up the configuration and test environment.

z/VM guest configuration

This is the configuration used for a z/VM guest.

The z/VM guests were interconnected by a z/VM guest LAN configured as type HiperSockets, with a maximum frame size (MFS) of 24 KB. All guests ran SLES 10 SP1, kernel level 2.65-7.244-s390x. Table 2 on page 7 gives details about the z/VM guest configuration.

Table 2. z/VM guest configuration

Host name	IP address	Number of virtual CPUs	Memory size	Function
LNX00080	192.168.30.10 (Internal zone)	2	2 GB	WebSphere Application Server 1
LNX00081	192.168.30.11 (Internal zone)	2	2 GB	WebSphere Application Server 2
LNX00082	192.168.30.12 (Internal zone)	2	2 GB	WebSphere Application Server 3
LNX00083	192.168.30.13 (Internal zone)	2	2 GB	WebSphere Application Server 4
LNX00084	192.168.30.100 (Internal zone)	1	512 MB	Apache HTTP server
LNX00086	192.168.30.20 (Internal zone)	1	2 GB	DB2 Server
LNX00090	192.168.30.21 (Internal zone) 192.168.40.21 (DMZ)	1	512 MB	Firewall 1
LNX00085	192.168.40.60 (DMZ)	1	512 MB	Caching Proxy Server or Proxy Server
LNX00091	192.168.40.22 (DMZ) 10.10.60.22 (OSA)	1	512 MB	Firewall 2

Test environment

The test environment consisted of an IBM System z and an IBM xSeries® server as the client.

The IBM System z contained a z/VM LPAR with seven guests. The network was split into three parts:

- The xSeries and the IBM System z systems were connected in the unsecure external zone through an Extreme Mariner Switch. The IBM xSeries system contained the WebSphere Studio Workload Simulator, which drove the Trade clients and generated the workload.
- The DMZ contained one guest with the WebSphere Caching Proxy Server protected from the external zone with a firewall (Firewall 2) running in a separate guest. It is implemented as a z/VM guest LAN. For the test scenario where the Caching Proxy Server and the WebSphere Proxy Server are bypassed, the Apache Web Server was moved to the DMZ. This was for testing purposes and is not recommended for production environments.
- The trusted internal zone, the second z/VM guest LAN, is protected with another guest with a second firewall (Firewall 1) and contains one guest for each of the following servers:
  - The Apache Web Server
  - The WebSphere Application Server Cluster (four guests)
  - The DB2 Universal Database (DB2 UDB) database server

Figure 1 on page 8 illustrates the test environment.

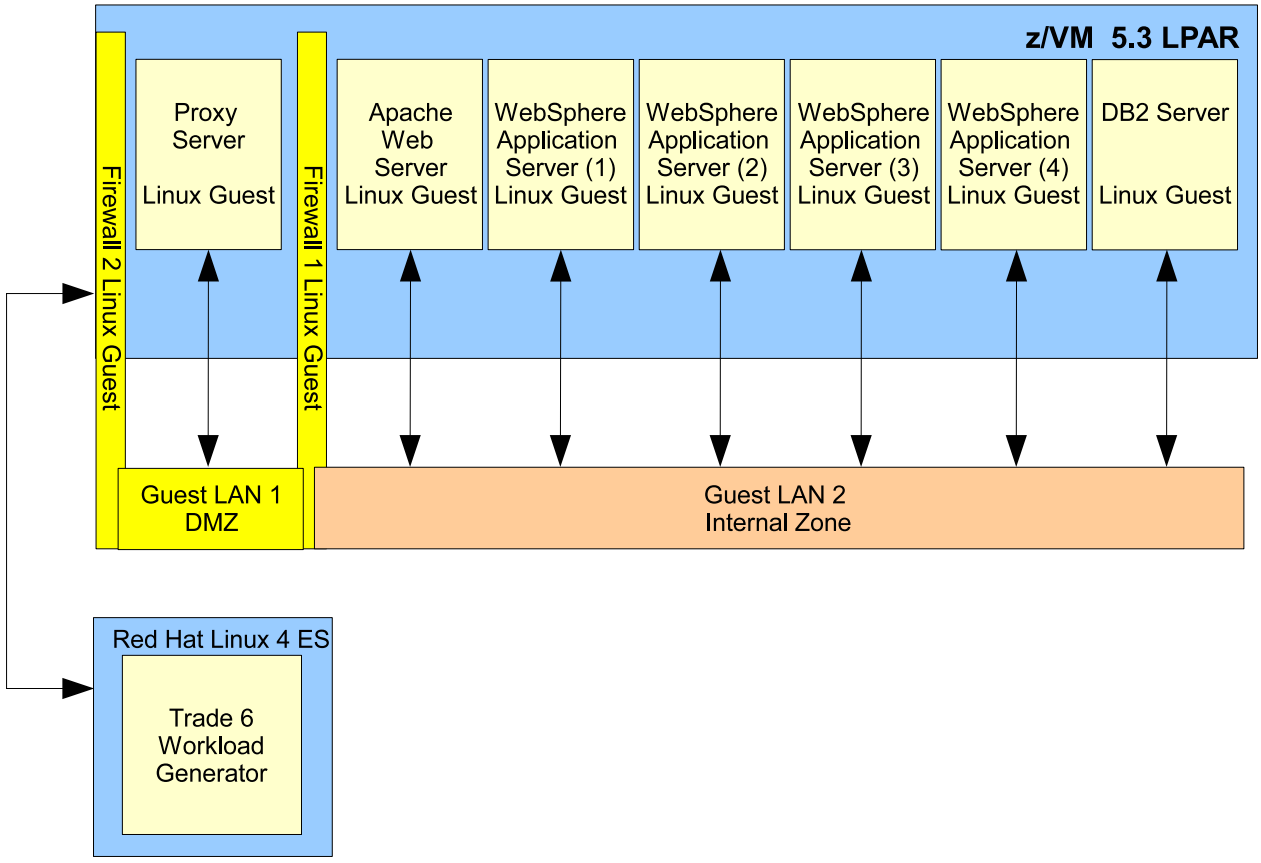

Figure 1. Test environment

Firewall Rules

These are the rules used to govern the two firewalls used in this study.

The firewall rules were structured in the following three areas:

- Incoming traffic
- Forwarding
- Outgoing traffic

The strategy was to deny most everything at first, and then allow some dedicated connections. The iptables rules were set up to allow **ping** and SSH commands. In a production environment, **ping** (ICMP) and SSH (TCP port 22) would probably be denied.

Firewall 1

These rules were used for Firewall 1:

Incoming traffic

1. Stop all incoming traffic.
2. Allow all related and established traffic for Firewall 1.

Forwarding traffic

1. Stop all forwarding traffic.

2. Allow forwarding of TCP traffic from 192.168.40.60 (proxy server) to the internal servers.
3. Allow forwarding of all related and established traffic.

Outgoing traffic

Allow output traffic for ICMP.

Note: This rule is for maintenance only and would probably not be implemented in a production environment.

All servers on the internal zone have Firewall 1 as their default route.

Firewall 2

These rules were used for Firewall 2:

Incoming traffic

1. Stop all incoming traffic.
2. Allow all related and established traffic for Firewall 2.

Forwarding traffic

1. Stop all forwarding traffic.
2. Allow forwarding of all related and established traffic.
3. Allow forwarding of TCP traffic on IP interface 10.10.60.0 (OSA card) to go to 192.168.40.21 (Firewall 1) and 192.168.40.60 (proxy server), and when Apache is moved into the DMZ, to 192.168.40.100.

Outgoing traffic

Allow output traffic for ICMP.

Note: This rule is for maintenance only and would probably not be implemented in a production environment.

The client needs to be able to route request through the Firewall 2 to the proxy server. The client routing is shown below.

```
[root@client ~]# route
Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
10.10.80.0      *               255.255.255.0   U     0      0        0 eth0
192.168.1.0     *               255.255.255.0   U     0      0        0 eth1
9.12.22.0       *               255.255.255.0   U     0      0        0 eth2
10.10.60.0      *               255.255.255.0   U     0      0        0 eth0
10.10.10.0      *               255.255.255.0   U     0      0        0 eth0
192.168.40.0    10.10.60.22     255.255.255.0   UG    1      0        0 eth0
169.254.0.0     *               255.255.0.0     U     0      0        0 eth2
default         pdlrouter-if7.p 0.0.0.0         UG    0      0        0 eth2
[root@client ~]#
```

IP address 10.10.10.60.22 (in **bold** in the above example) is the address assigned to an OSA adapter configured on the Firewall 2 z/VM guest.

To enable the Firewall 2 to route IP traffic on this OSA adapter, the adapter must be configured as a primary router. The address of the OSA adapter on Firewall 2 is 0716 through 0718.

To enable route4 primary routing, the configuration file for the OSA adapter needed to be changed to include a QETH_OPTIONS='route4=primary_router' clause (shown in **bold** in the following example).

```
firewall2:/etc/sysconfig/network # cat /etc/sysconfig/hardware/hwcfg-qeth-bus-ccw-0.0.0716
CCW_CHAN_IDS='0.0.0716 0.0.0717 0.0.0718'
CCW_CHAN_MODE=''
CCW_CHAN_NUM='3'
LCS_LANCMD_TIMEOUT=''
MODULE='qeth'
MODULE_OPTIONS=''
QETH_IPA_TAKEOVER='0'
QETH_LAYER2_SUPPORT='0'
QETH_OPTIONS='route4=primary_router'
SCRIPTDOWN='hwdown-ccw'
SCRIPTUP='hwup-ccw'
SCRIPTUP_ccw='hwup-ccw'
SCRIPTUP_ccwgroup='hwup-qeth'
STARTMODE='auto'
firewall2:~ #
```

Workload description

The Trade performance benchmark is a workload developed by IBM for characterizing performance of the WebSphere Application Server.

The workload consists of an end-to-end Web application and a full set of primitives. The applications are a collection of Java™ classes, Java Servlets, Java Server Pages, Web Services, and Enterprise JavaBeans built to open J2EE 1.4 APIs. Together, these provide versatile and portable test cases designed to measure aspects of scalability and performance. Figure 2 shows an overview of the Trade J2EE components.

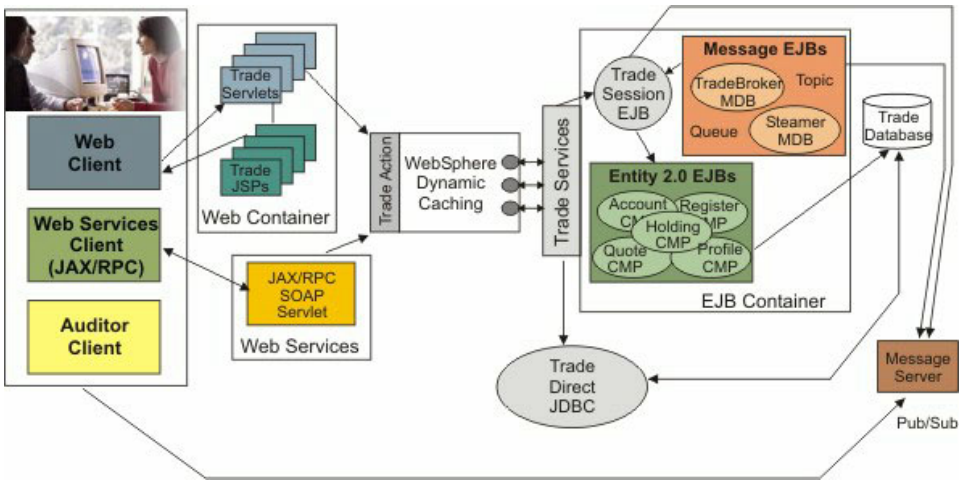

Figure 2. Trade J2EE components

The new Trade benchmark (Trade 6) has been redesigned and developed to cover WebSphere's significantly expanding programming model. This provides a more realistic workload driving WebSphere's implementation of J2EE 1.4 and Web services including key WebSphere performance components and features.

Trade's new design spans J2EE 1.4 including

- The new EJB 2.1 component architecture
- Message Driven beans (MDBs)
- Transactions (1-phase, 2-phase commit)

- Web Services (SOAP, WSDL)

Trade also highlights key WebSphere performance components such as DynaCache, WebSphere Edge Server, and Web Services.

Trade is modeled after an online stock brokerage. The workload provides a set of user services such as login/logout, stock quotes, buy, sell, account details, and so on, through standards-based HTTP and Web services protocols.

Trade provides the following server implementations of the emulated Trade brokerage services:

EJB Database access uses EJB 2.1 technology to drive transactional trading operations.

Direct Database and messaging access through direct JDBC and JMS code.

Type four JDBC connectors were used with EJB containers. See Figure 2 on page 10 for details.

EJB 2.1

Trade 6 continues to use the following features of EJB 2.0 and leverages EJB 2.1 features such as enhanced Enterprise JavaBeans™ Query Language (EJBQL), enterprise Web services and messaging destinations.

Container-Managed Relationships
One-to-one, one-to-many, and many-to-many object to relational data managed by the EJB container and defined by an abstract persistence schema. This feature provides an extended, real-world data model with foreign key relationships, cascaded updates and deletes, and so on.

EJBQL
Standardized, portable query language for EJB finder and select methods with container-managed persistence.

Local and Remote Interfaces
Optimized local interfaces providing pass-by reference objects and reduced security overhead.

The WebSphere Application Server provides significant features to optimize the performance of EJB 2.1 workloads. Trade uses access intent optimization to ensure data integrity, while supporting the highest performing and scalable implementation. Using access intent optimizations, entity bean runtime data access characteristics can be configured to improve database access efficiency, which includes access type, concurrency control, read ahead, collection scope, and so forth.

The J2EE programming model provides managed, object-based EJB components. The EJB container provides declarative services for these components such as persistence, transactions, and security. The J2EE programming model also supports low-level APIs such as JDBC and JMS. These APIs provide direct access to resource managers such as database and message servers. Trade provides a Direct implementation of the server-side trading services using direct JDBC. This implementation provides a comparison point to container-managed services that details the performance overhead and opportunity associated with the EJB container implementation in WebSphere Application Server.

Note:

- All the measurements done in this study used EJB.
- Most enterprises now use some form of ORM tool to manage data kept in the database.

Trade provides two order processing modes: asynchronous and synchronous. The order processing mode determines the mode for completing stock purchase and sell operations. Asynchronous mode uses MDB and JMS to queue the order to a TradeBroker agent to complete the order. Asynchronous_2-Phase performs a two-phase commit over the EJB database and messaging transactions. Synchronous mode, on the other hand, completes the order immediately.

Note: All the measurements done in this study used synchronous order processing mode.

Trade provides the following access modes to the server-side brokerage services.

Standard
Servlets access the Trade enterprise beans through the standard RMI protocol.

WebServices
Servlets access Trade services through the Web services implementation in WebSphere Application Server. Each trading service is available as a standard Web service through the SOAP Remote Procedure Call (RPC) protocol. Because Trade is wrapped to provide SOAP services, each Trade operation (login, quote, buy, and so on) is available as a SOAP service.

Note:
- All the measurements done in this study used the Standard access mode.
- For all measurements in this study, the Trade database was populated with 2000 users (uid:0 through uid:2000) and 1000 quotes (s:0 through s:999).

Trade can run with any of the following WebSphere Application Server caching modes.

No cache
No caching is used.

Command caching
This caching feature was added to WebSphere Application Server V5.0 for storing command beans in the dynamic cache service. Support for this feature was added in Trade 3 and carried over to Trade 6.

Note: Servlet caching can also be enabled with command caching.

Distributed Map
This feature is new in WebSphere Application Server V6.0, providing a general API for storing objects in the dynamic cache service.

Note: For these tests, all of the above caching modes were examined.

In the test environment, Trade requests followed a particular path. When a Trade client issued a request, the request was routed through the first firewall, (Firewall 2 in Figure 1 on page 8), to the caching proxy server, which was allowed to forward the request through the back firewall, (Firewall 1 in Figure 1 on page 8), to the WebSphere Load Balancer. The WebSphere Load Balancer then forwarded the request to the Web server, which in turn, forwarded it to WebSphere Application Server. Communication between the WebSphere Load Balancer, the Web server, the

WebSphere Application Servers, and the database occurred over the internal z/VM guest LAN (Guest LAN 2 in Figure 1 on page 8. Responses back to the client followed the same path, in reverse.

Detailed information about the Trade workload can be found in the IBM Redbooks® publication "Using WebSphere Extended Deployment V6.0 To Build an On Demand Production Environment" at http://www.redbooks.ibm.com/abstracts/sg247153.html

Information on how Trade works with the WebSphere Application Server can be found at http://www.ibm.com/software/webservers/appserv/was/performance.html

WebSphere Studio Workload Simulator

The Trade workload was driven by the WebSphere Studio Workload Simulator. Clients are simulated by worker threads. Increasing the number of clients increases the transaction rate or load on the Trade application running on the WebSphere Application Server. Both Command caching and Distributed Map caching were used.

Caching overview

The following summary is taken from the IBM Redbooks publication 'Mastering DynaCache in WebSphere Commerce'.

http://www.redbooks.ibm.com/abstracts/sg247393.html

Where caching is performed

There are several places where caching is performed.

In a typical IBM WebSphere topology, caching can be performed at several places. Some of the most notable caching locations are:

- At the Web client and browser.
- At the Internet Service Provider (Akamai is an example).
- In a caching proxy server located in front of the application server.
- In the HTTP Web server (for example, static content and edge side includes).
- At the application server in DynaCache.
- In the back-end database caching buffer pools.

Client-side caching

Caching capabilities are built in to most Web browsers today, and in that case, the cache works only for a single user. For example, the browser checks if a local copy of a Web page is available and if this is true, the time stamp of the local copy in the browser cache is recorded.

This time stamp will be sent to the Web server in the following HTTP GET request. The browser might request the Web page by specifying the requested URI as "/". In that same Web page request, the browser can use the HTTP header request field If-Modified-Since to indicate to the Web server that it already has a cached version that is time stamped "Sat, 10 July 200X 10:00:00 GMT."

The Web server checks the page modification time against the time specified in the HTTP request. The Web server determines that the page has not been modified

since ″Thurs, 6 May 200X 09:50:00 GMT,″ so it replies back to the browser that the page has not been modified. A return HTTP 304 response code is used to notify that a page has not changed since the specified date.

In this example, the Web server has indicated that the page has not been modified, so the browser cache entry is current. Therefore the browser displays the page from the cache. In this case, the browser also assumes that none of the images contained in the page has been modified.

Server-side caching

When caching systems are implemented between the client and the application server, they are known as proxy servers or proxy caches. Ideally, caches are placed as close to the client as possible without compromising security. DynaCache is an example of a server-side caching mechanism.

Proxy server caches are strategically placed near network gateways in order to reduce traffic, increase network bandwidth, and decrease the overall costs of network connections. A single proxy server can easily manage many users simultaneously while maintaining cached objects derived from many sources.

Most of the benefits are derived from caching objects requested by one client for later retrieval by another client. Several proxy servers can also be joined together into a cluster or hierarchy such that any cache can request items from a neighboring cache member, the assumption being that doing so reducesthe need to fetch the object directly from the source of origin.

Full Web page versus fragment caching

A fragment is a part or all of a rendered HTML page. Although a whole page may not be *cacheable*, it may contain sections of varying *cacheability* that can be separated into fragments and then cached independently. Any content that can be independently requested can be cached. This means that a fragment cannot depend on any information placed in its request scope by its parent or from other fragments. That means, if the parent object gets invalidated, the child object can be executed on its own. In the test environment, the Trade application does not make use of page fragments.

Static versus dynamic object caching

Most caching strategies target Web site content that rarely changes, such as graphical images and text files. However, many sites serve dynamic content, containing personalized information and data that change more frequently.

Caching dynamic content requires more sophisticated caching techniques. The IBM WebSphere Application Server DynaCache system provides an elegant solution in terms of caching dynamic content.

DynaCache technical overview

Think of DynaCache as a sophisticated Java Hashtable.

The code used to provide the in-memory cache services extends the Java Dictionary class, which is the abstract parent of Hashtable. You configure DynaCache to store objects, and later, based on some data matching rules, DynaCache retrieves those objects and serves them from its cache. Caching rules

are stored in a configuration file called cachespec.xml. Single or multiple caches are supported. Caches are stored in the JVM heap memory and DynaCache supports overflow to disk if enabled and when required.

The system administrator has some degree of control over what is placed in memory and what (if anything) overflows to disk via configuration settings. DynaCache also calls a least recently used (LRU) algorithm during the item selection process for memory-based item eviction or overflow. The LRU algorithm consults with a user-specified cache entry priority number before the evictee is chosen. Higher numbered items stay in memory longer.

Disk offload of cache entries from memory occurs when the memory cache fills up or when the server is in the process of performing a normal, administrator-directed shut down and the ″FlushToDiskOnStop″ property is enabled. Upon a server restart, requests for cache entries are fulfilled by reloading the saved disk data into memory.

Note: On server startup DynaCache does not load the entire disk cache into memory.

DynaCache removes stale cache items, both as individuals or dependent, related groups. The process of removing these items is called invalidation. DynaCache creates a user-defined, unique ″key″ to store and retrieve cache items and a second, optional, shared group key for group invalidation. DynaCache also provides an API for developers to call invalidation as a runtime function.

DynaCache is supported in WebSphere clustered environments using the Distributed Replication Service (DRS).

Applications running on an application server can access cache instances on other application servers as long as they are part of the same replication domain. WebSphere Application Server V5.1 DynaCache provided a feature called cache instance. In V6, this feature was extended and now provides two types of cache instances: servlet cache instance and object cache instance. Each cache instance is independent of and not affected by any other cache instances.

The servlet cache instance stores servlets, JSPs, Struts, Tiles, command objects and SOAP requests. It allows applications such as WebSphere Portal Server and WebSphere Application Server to store data in separate caches. In the test environment, the Trade command caching mode uses this cache instance.

The object cache instance is used to store, distribute, and share Java objects. The DistributedObjectCache and Distributed Map APIs are provided so the applications can interact with the object cache instances. In the test environment, the Trade distribute map caching mode uses this cache instance.

The Distributed Map and DistributedObjectCache interfaces are simple interfaces for the DynaCache. Using these interfaces, J2EE applications and system components can cache and share Java objects by storing a reference to the object in the cache. The default DynaCache instance is created if the DynaCache service is enabled in the Administrative Console. This default instance is bound to the global Java Naming and Directory Interface™ (JNDI) namespace using the name services/cache/distributedmap. Figure 3 on page 16 illustrates how DynaCache works.

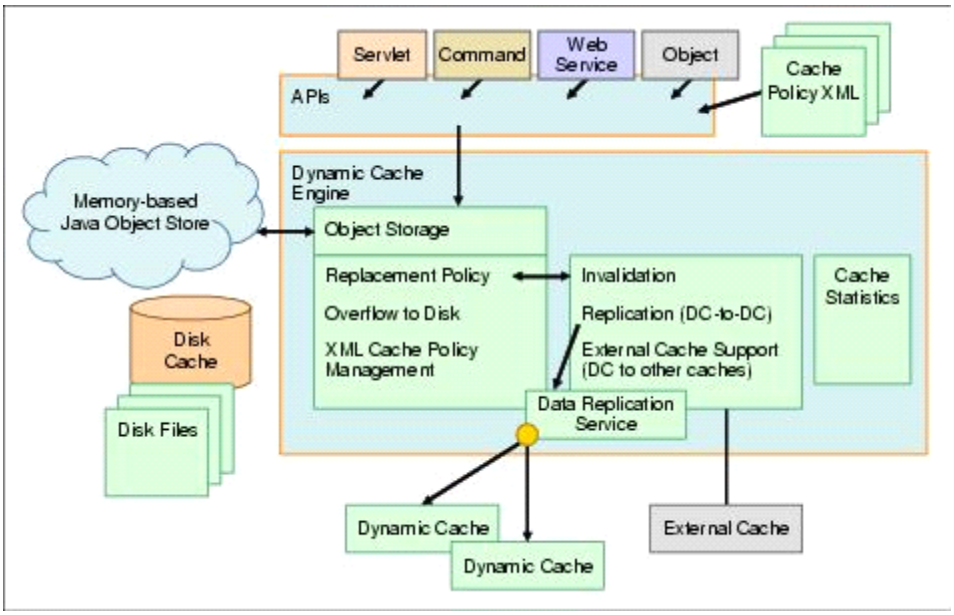

Figure 3. WebSphere Application Server DynaCache overview

The DynaCache service works within an application server Java Virtual Machine (JVM), intercepting calls to cacheable objects. For example, it intercepts calls to the servlet service() method or a Java command performExecute() method, and either stores the output of the object to the cache or serves the content of the object from the DynaCache. For a servlet, the resulting cache entry contains the output or the side effects of the invocation, such as calls to other servlets or JSP files, or both.

The DynaCache loads and builds a caching policy for each cacheable object from its configuration cachespec.xml file located under the WEB-INF directory. This policy defines a set of rules specifying when and how to cache an object (that is, based on certain parameters and arguments), and how to set up dependency relationships for individual or group removal of entries in the cache.

Each data request (meaning any invocation of a cacheable servlet, JSP, Web service, or other object) is associated with a set of input parameters that are combined to form a unique key, called a cache identifier or cache-id. A key policy defines which cache identifiers result in a cacheable response. If a subsequent request generates the same key, the response is served from the cache. If a unique key does not match any of the rules, the response for the request is not cached.

Features of DynaCache

DynaCache provides support for caching the returned results from Java object calls.

The main features of DynaCache are described in Table 3.

Table 3. Main features of DynaCache

Cache object	Description
Servlet/JSP results	DynaCache can cache any existing whole page or fragment generated by a servlet or JSP. DynaCache matches data found in the HTTP request header with the cache rules specified in the XML configuration file, named cachespec.xml.

Table 3. Main features of DynaCache (continued)

Cache object	Description
Command cache (Web Services, POJOs and EJBs)	Used to cache dynamic data before it is transformed into a presentable format (that is, HTML). These include EJB and Web service responses. A cacheable command object must inherit from the class com.ibm.websphere.command.cachableCommandImpl for it to work with DynaCache.
Replication support	Enables cache sharing and replication in a clustered environment.
Invalidation support	Includes rules-based, time-based, group-based, and programmatic cache invalidation techniques to ensure the freshness, consistency, and accuracy of the content.
Edge of network caching support	Extends the WebSphere Application Server caches into network-based caches, through the implementation of external cache control and distributed-fragment caching and assembly support.
JSR168 compliant portlets	DynaCache can cache any existing whole page or fragment generated by a servlet, JSP or a JSR 168 compliant Portlet1. DynaCache matches data found in the HTTP request header with the cache rules specified in the XML configuration file, named cachespec.xml.
Cache monitor	Cached results can be viewed and managed using a cache monitor J2EE application. The cache monitor provides a GUI interface to change the cached data manually. You have to install the application manually with the CacheMonitor.ear file (located in the <WAS_HOME>/ installableapps directory) using the WAS Admin GUI. This installable Web application provides a real-time view of the state of the DynaCache. The only way to externally manipulate the data in the cache is by using the cache monitor. After installation, the cache monitor can be accessed as: http://your host_name:your port_number/cachemonitor.

The IBM Extended Cache Monitor for IBM WebSphere Application Server provides two functions that are not available with the cache monitor that is supplied with WebSphere Application Server. These two functions enable you to:

- Display the contents of object cache instances.
- Display the Dynamic Cache mbean statistics for cache instances.

The IBM Extended Cache Monitor for IBM WebSphere Application Server is available from http://www.ibm.com/developerworks/websphere/downloads/cache_monitor.html.

Caching servlets and JSPs

The process of caching servlets and JSPs is explained and illustrated.

When the application server starts and a targeted servlet or JSP is called for the first time, no cached entry is found and so the service() method of the servlet is called. The DynaCache Web container intercepts the response from the service() invocation and caches the results. This process is illustrated in Figure 4 on page 18.

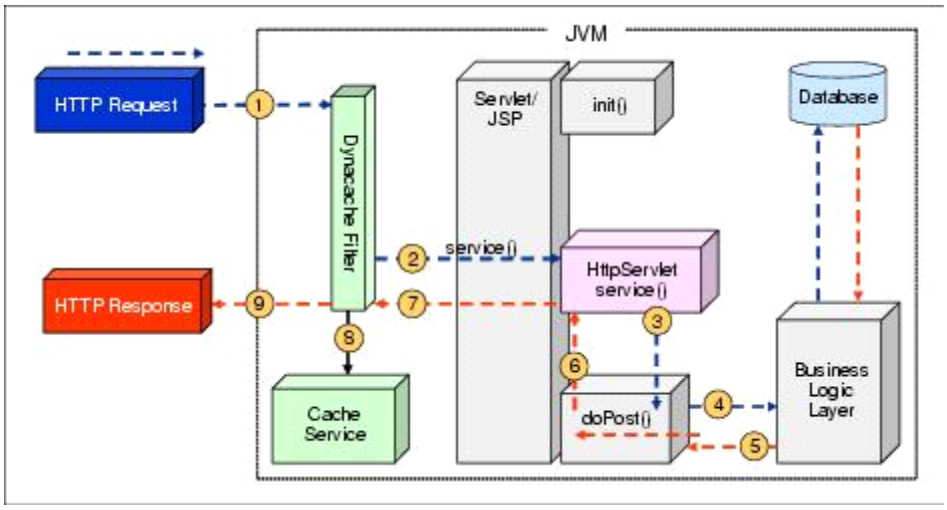

Figure 4. Before caching: A request must traverse all layers

The next time that the servlet is called and a match is found in the caching rules engine, the cached results are returned and the service() method is not called. This bypasses all of the processing that would have been done by the servlet, resulting in a substantial performance boost.

If, however, there is no cache entry for this ID, the service() method is run as normal, and the results are caught and placed in the cache before they are returned to the requester.

Figure 5 illustrates how DynaCache increases performance by bypassing the service() method call entirely whenever a cache hit occurs.

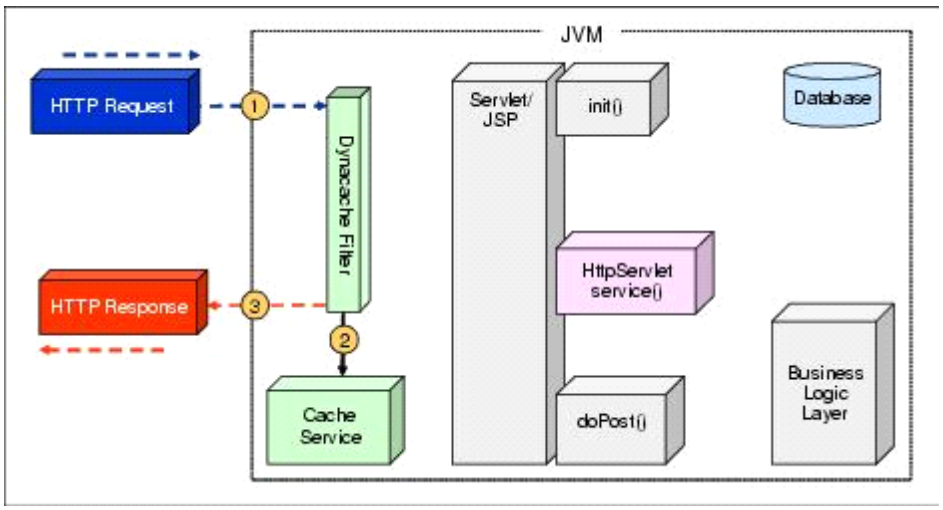

Figure 5. A cache hit means that the servlet and business logic is not called

Servlets and JSPs are configured for caching with entries in the file cachespec.xml.

Java objects and the command cache

DynaCache provides support for caching the returned results from Java object calls. DynaCache provides a very straightforward mechanism that allows you to cache the results from these Java command objects; it is called command caching.

Commands written to the WebSphere command architecture access databases, file systems, and connectors to perform business logic, and often run remotely on another server. Significant performance gains can be achieved by caching command results and avoiding their repeated invocation.

To take advantage of command caching, applications must be written to the WebSphere command framework API. This API is based on a common programming model, and uses a series of getters(), setters() and execute() methods to retrieve its results. To use a command, the client creates an instance of the command and calls the execute() method for the command. Depending on the command, calling other methods could be necessary. The specifics will vary with the application.

With Command cache, the data is cached before it is transformed into HTML. In order to take advantage of command caching, you must make simple code changes to your Java objects so that DynaCache can call them to retrieve the data. Command objects inherit from com.ibm.websphere.command.cachableCommandImpl and must provide certain methods to function properly.

Commands are Java objects that follow a usage pattern that consists of three operational methods. They are:

Set Initialize the input properties of the command object.

Execute
Perform the specific business logic for which the command was written.

Get Retrieve the output properties that are set by the command execution.

Each command can be in one of three states based on which methods have been executed:

New The command has been created but the input properties have not been set.

Initialized
The input properties have been set.

Executed
The execute method has been called and the output properties have been set and can be retrieved.

Executed command objects can be stored in the cache so that subsequent instances of that command object's execute method can be served by copying output properties of the cached command to the current command for retrieval by its get methods. DynaCache supports caching of command objects for later reuse by servlets, JSPs, EJBs, or other business logic programming.

Figure 6 on page 20 shows the logic of a simple stock quote function similar to the Trade application.

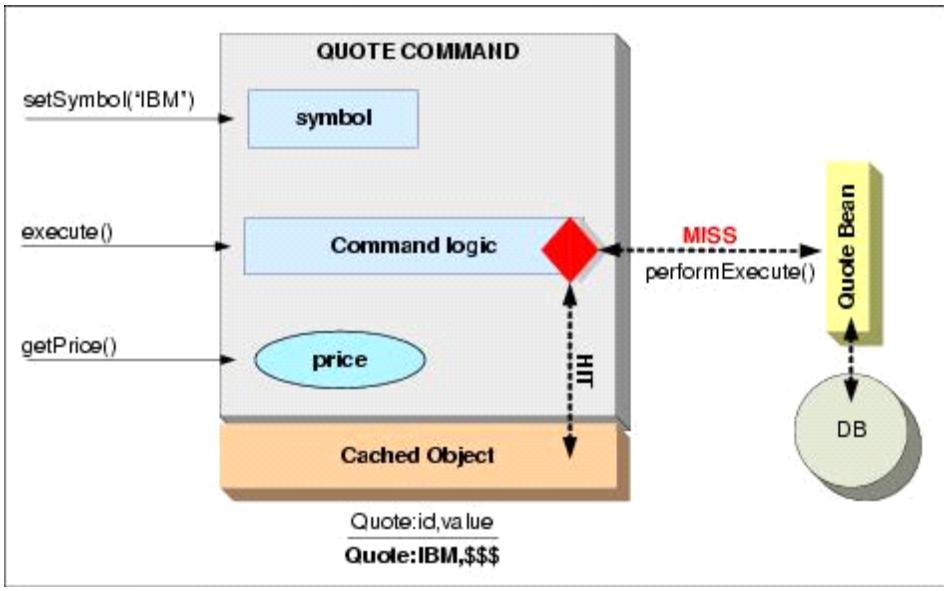

Figure 6. Implementing a quote command used in a stock application

Data replication service

The Data Replication Service (DRS) is an internal, proprietary component of the WebSphere Application Server. The DRS is used by other components to move data from node to node within a network of clustered application servers.

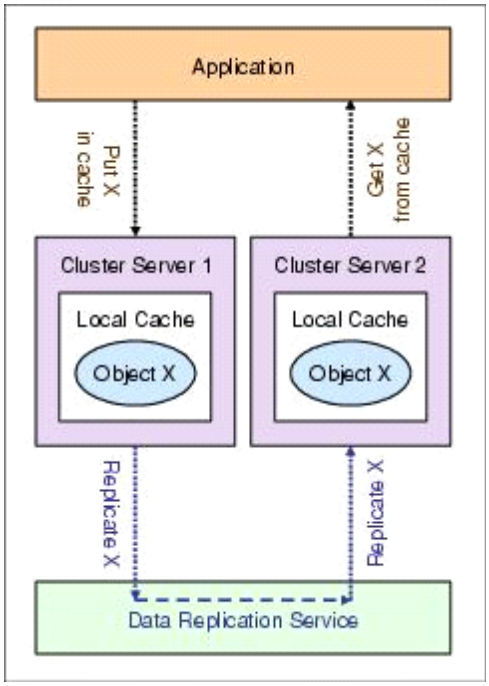

Figure 7. Moving a cache entry across cluster members using DRS

In Figure 7, DRS is used to share data between the four cluster members, numbered 1 through 4.

Figure 8 illustrates a WebSphere cluster. The workload is balanced between two IBM HTTP servers, which take input from all four WebSphere cluster members.

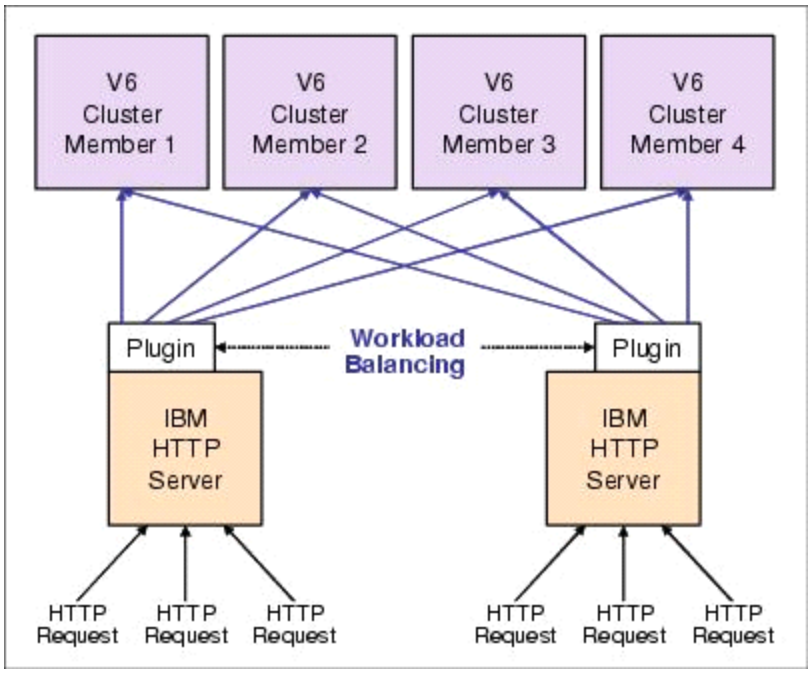

Figure 8. Example of a clustered WebSphere topology

The Data Replication Service caters to two scenarios: failover and caching. This study analyzes the caching feature; failover scenarios were not analyzed.

When an object is available in the cache, repeated requests for the same information are handled faster. Because the data in the cache must be the same, irrespective of which application server the request arrived at, it makes sense to cache the same data in of all the application servers. Data Replication Service handles moving cache entries from server to server.

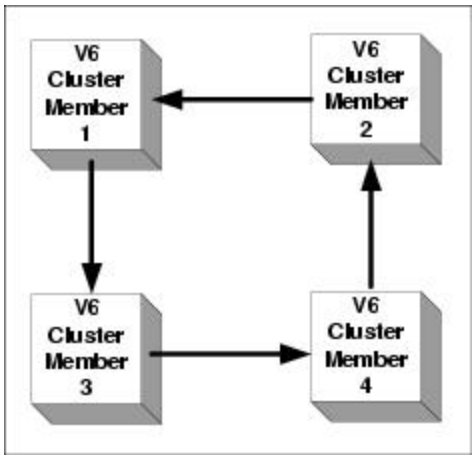

Figure 9a. Default topology configuration for V6 Distributed Replication Services (DRS)

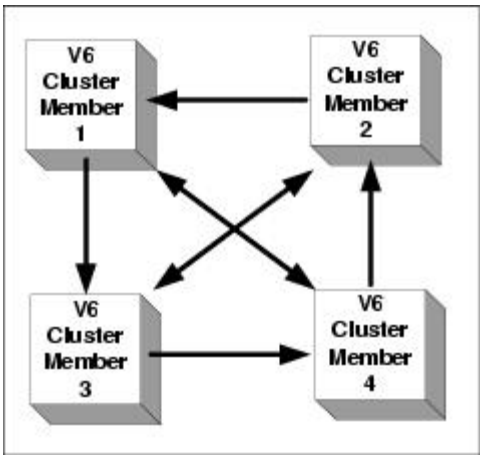

Figure 9b. Topology required for DynaCache DRS to function properly

Figure 9. Topology configurations

In the default topology, (left side of Figure 9), each server in the domain holds a replica of the data from one other server. In the second example, (right side of Figure 9), the double headed arrows mean that data flows from each process to every other process, so that for each replicated object, there are three remote copies and the local original. The type of topology is specified when the replication domain is created. Then, the entire domain selection corresponds to this topology.

This topology would probably be more than what is needed for HTTP session replication. In the DynaCache environment, the topology described in the right side of Figure 9, is the only allowable configuration for cache replication because when caching dynamic content, the cache is only useful if it is available on all the machines where a request could arrive.

This study used only the caching capability with the topology as shown in the right side of Figure 9.

Setting up and configuring caching in the test environment

This section describes the two basic types of caching content, static and dynamic.

Static content

This content does not change over long periods of time. The content type can be HTML, JSP rendering less dynamic data, GIF, JPG, or others. Static content is characterized by content that is not generated dynamically and has predictable expiration values.

Static content typically resides on the file system of the Web server or application server and is served unchanged to the client. To a large extent, this content can be declared to be cacheable. This type of content is typically cached at a caching proxy. See Figure 10 on page 23.

Dynamic content

This content includes frequently updated content (such as exchange rates), as well as personalized and customized content. Although it is changing content, at the same time it must be stable over a long enough period of

time for meaningful reuse to occur. However, if some content is very frequently accessed, such as requests for pricing information of a popular stock, even a short period of stability may be long enough to benefit from caching.

Dynamic content is typically cached in the dynamic cache services of the application server or the ESI cache of the Web server plug-in (note that these tests did not use ESI caching). See Figure 10.

Caching proxy server setup

The caching proxy server stores cacheable content in a local cache before delivering it to the requester. It does not store all content, only specific cacheable content that meets its rules.

You can configure the caching proxy server to handle protocols such as HTTP, FTP, and Gopher. Examples of cacheable content include static Web pages and whole dynamic Web pages. Caching enables the caching proxy server to satisfy subsequent requests for the same content by delivering it directly from the local cache, which is much quicker than retrieving it again from the content host. For these test, the caching proxy server was configured to use memory.

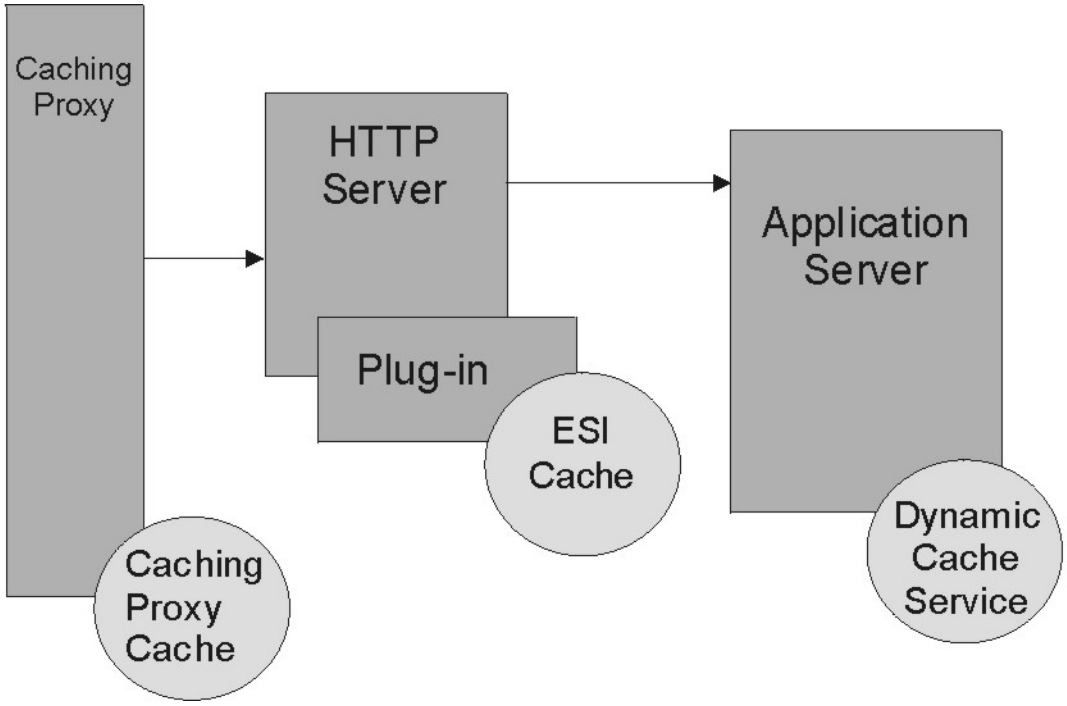

Figure 10. Caching content types

Dynamic caching with Trade and WebSphere

The dynamic cache service improves performance by caching the output of servlets, commands, and Java Server Pages (JSP) files.

By caching the response from servlets, Web services, JSPs, and WebSphere Application Server commands, the application server does not have to perform the same computations and backend queries multiple times.

The WebSphere Application Server consolidates several caching activities, including servlets, Web services, and WebSphere commands into one service called the

dynamic cache. These caching activities work together to improve application performance and share many configuration parameters, which are set in an application server's dynamic cache service.

The dynamic cache works within an application server Java Virtual Machine (JVM) by intercepting calls to cacheable objects. For example, calls through a servlet's service() method or a command's execute() method. The dynamic cache then either stores the object's output to the dynamic cache, or serves the object's content from the dynamic cache. J2EE applications are likely to have high read/write ratios and can tolerate a small degree of latency.

To make the data current, the dynamic cache can create an opportunity for significant gains in server response time, throughput, and scalability. However, the possible use of dynamic caching technology should be reviewed with application developers and owners. This is so that they will design and program to ensure that data validity is maintained, should WebSphere data caching be activated.

A cache hit is generally served in a fraction of the time of a non-cached request. Significant performance gains are generally achieved with Trade when dynamic caching technology is used.

The parameters that can be set in the WebSphere Application Server for the dynamic cache service are:

- The size of the cache, which is expressed as the maximum number of entries that the cache can hold. This value is set using the WebSphere Application Server administration console under **application servers → server_name → web container services → dynamic cache service**. The actual caching service mode is set by each application individually.
- The priority, disk cache setting, and Domain Replication Services (DRS).

WebSphere Application Server V6.0 provides two interfaces, DistributedObjectCache and Distributed Map. These applications are used by J2EE applications to cache and share Java objects by storing a reference to the object in the cache. These interfaces expand on the command bean and servlet fragment caching capabilities already provided in previous versions of WebSphere Application Server. By default, the caching capabilities within Trade are disabled. However, the caching type can be modified to use either the Distributed Map interface or the command bean caching (Command caching) interface.

Trade provides settings for multiple caching modes. By modifying the setting in the web.xml file, (located in directory `/opt/IBM/WebSphere/AppServer/profiles/default/installedApps/<WebSphere node>/trade.ear/tradeWeb.war/WEB-INF`), you specify the dynamic caching technology to be used by Trade. By default, data caching is disabled (set to No caching) in Trade. The Trade supported caching modes are:

No cache
: No caching is used.

Command caching
: This caching feature was added to WebSphere Application Server V5.0 for storing command beans in the dynamic cache service. Support for this feature was added in Trade 3 and carried over to Trade 6.

Distributed Map
: This feature is new in WebSphere Application Server V6.0, providing a general API for storing objects in the dynamic cache service.

For the measurements taken in this study, all three caching modes (No cache, Command Caching, and Distributed Map caching) were used. See web.xml in "web.xml setup examples" on page 61 for the section of the web.xml file to modify in order to use dynamic caching.

Dynamic caching requires that the application supply a cachspec.xml file (found in directory `/opt/IBM/WebSphere/AppServer/profiles/default/installedApps/<WebSphere node>/trade.ear/tradeWeb.war/WEB-INF`) that defines the caching policies for the application. File cachespec.xml (see "web.xml setup examples" on page 61) contains the file that was used for all measurements in these tests.

The cachespec.xml file specifies the caching behavior for command caching and servlet caching.

For Distributed Map caching, the application uses the Distributed Map API to control caching behavior.

Enabling WebSphere Application Server DynaCache

DynaCache is ready-to-go straight out of the box. It is available immediately after you have installed WebSphere Application Server, although it needs to be enabled by a simple administration switch setting. By default, the service is enabled.

See Figure 11 and Figure 12 on page 26 for the panels used to enable DynaCache Service.

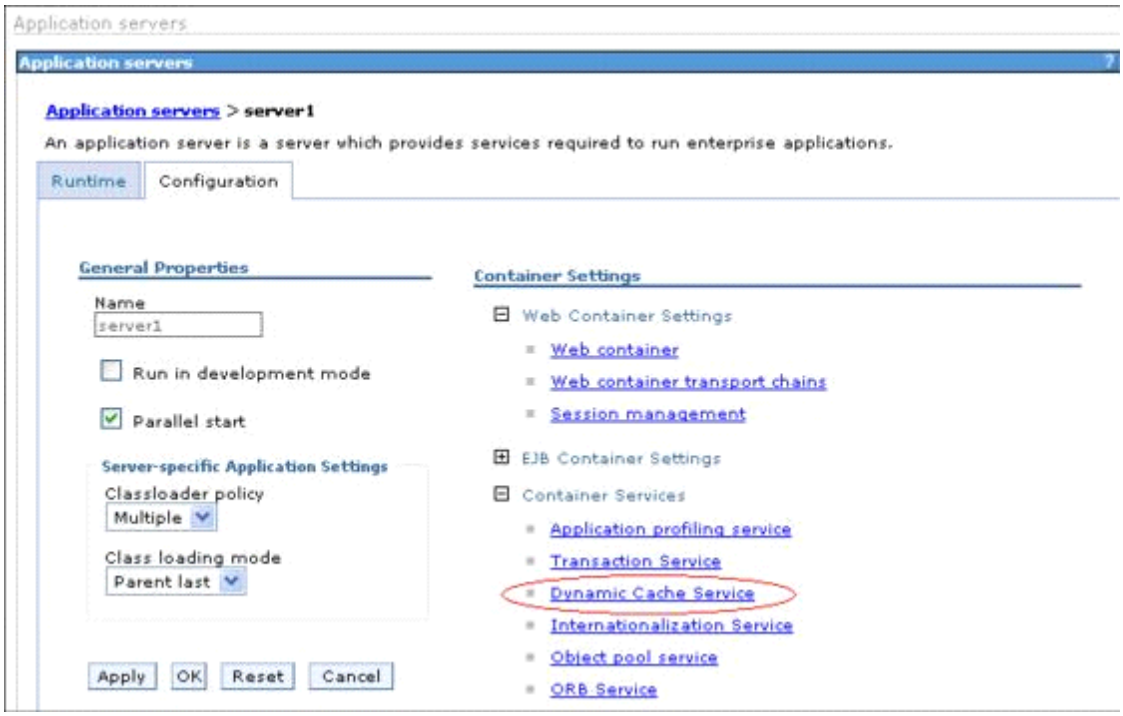

Figure 11. DynaCache Service in the administration console

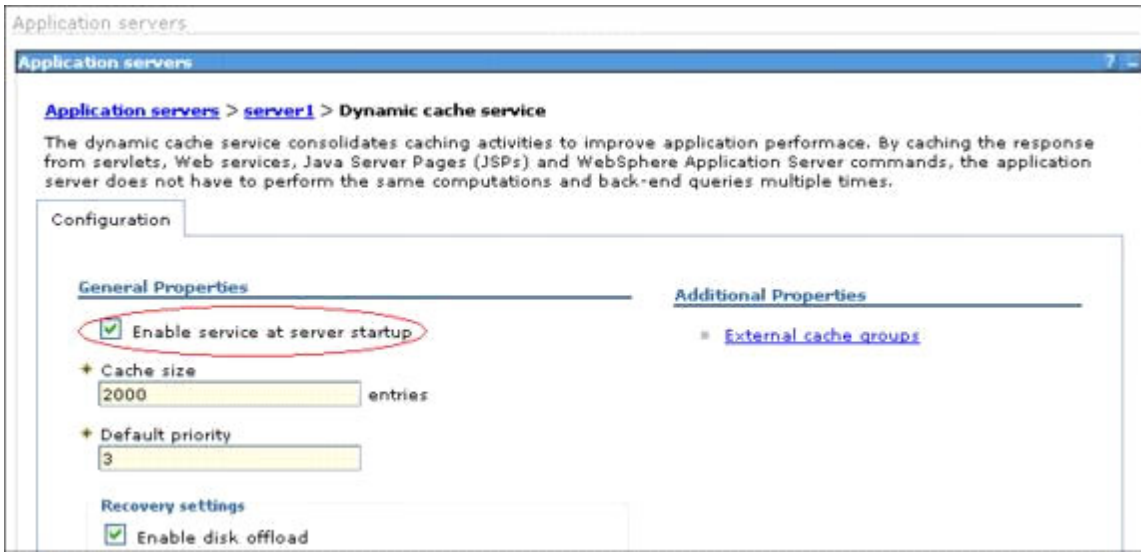

Figure 12. Enabling the dynamic caching service

CPU utilization charts explained

The charts in this publication that depict the CPU utilization values are always normalized in a way that 100% means that one CPU is fully utilized.

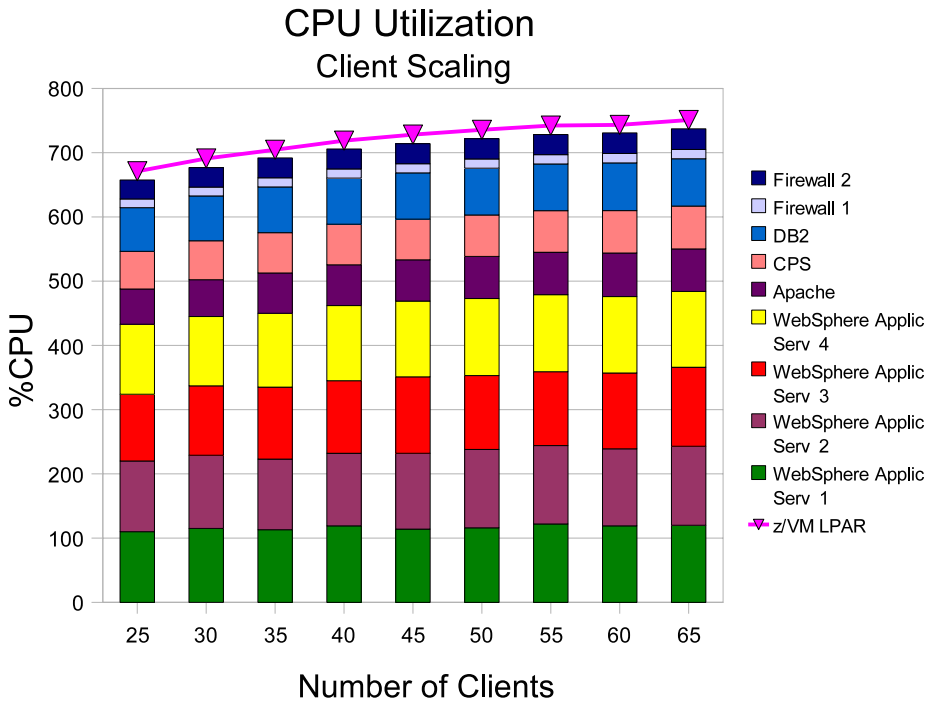

Figure 13. Example CPU utilization

In Figure 13, the CPU utilization charts show the utilization for the z/VM LPAR by a line with triangle symbols. The details of how the z/VM guests use the CPU is

shown with stacked bars, which should sum up to the z/VM utilization without the CP related CPU part, so it is always a little bit smaller than the z/VM LPAR load. If the CPU load from a guest in the legend is not visible, that means it is too small to display.

Test case scenarios and results

The following section provides a description of the test case scenarios that were used and the observed results for each scenario.

Test case scenarios

Unless otherwise noted, this is the standard test environment for the WebSphere Application Server tests.

Figure 14 illustrates the test environment. See "Test environment" on page 7 for more details.

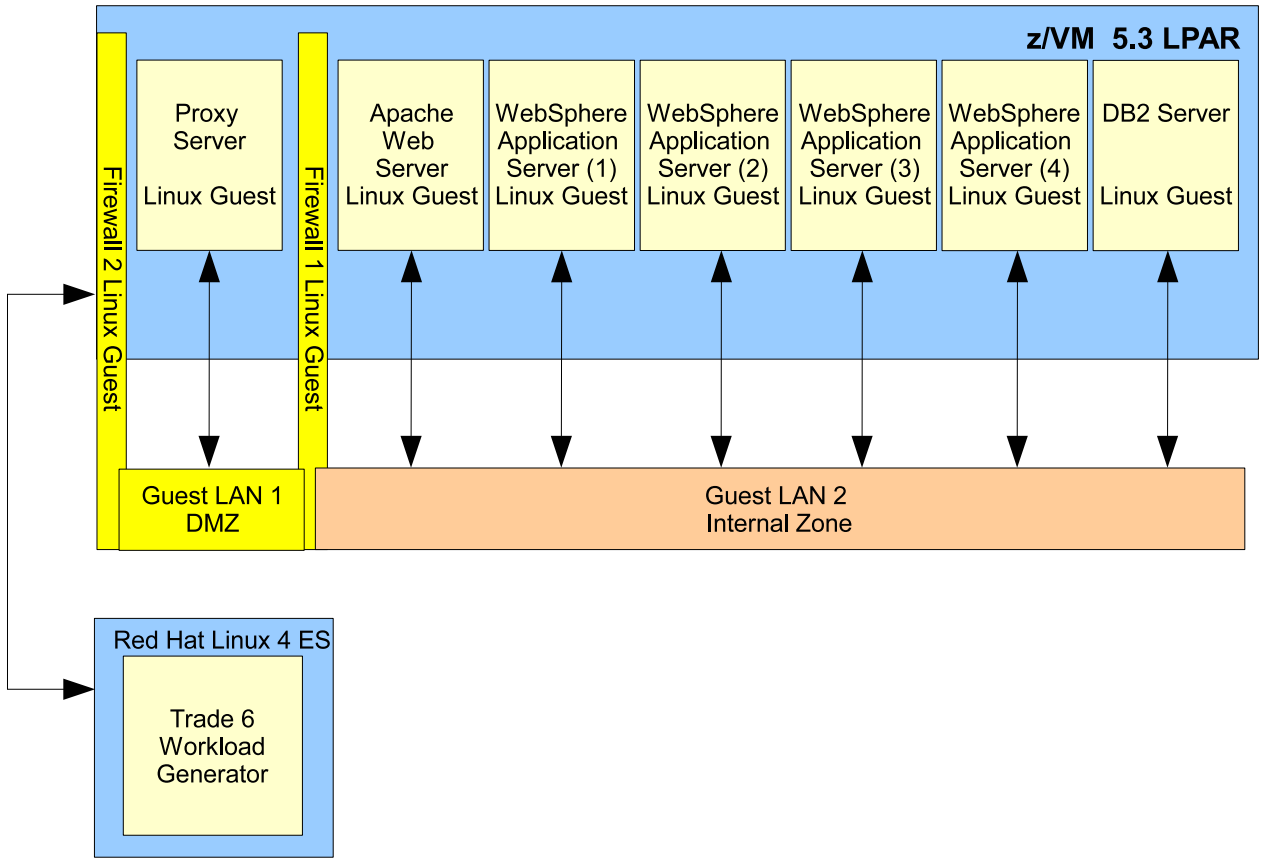

Figure 14. WebSphere Application Server test environment.

There were a total of four WebSphere Application Servers in the cluster, one workload simulator machine, one Apache Web Server and one WebSphere Edge Component Proxy Server or WebSphere Application Server Proxy Server node.

Table 4 shows the setup of the standard configuration.

Table 4. Setup of standard configuration for the WebSphere Application Server tests.

System/Guest	Memory size	Number of CPUs
z/VM	12288 MB	8
Caching Proxy Server	512 MB	1
Apache HTTP Server	512 MB	1
WebSphere Application Server 1	2 GB	2
WebSphere Application Server 2	2 GB	2
WebSphere Application Server 3	2 GB	2
WebSphere Application Server 4	2 GB	2
DB2 Server	4 GB	1
Firewall 1	512 MB	1
Firewall 2	512 MB	1

The Trade application was configured for 1000 stock quotes and 2000 users.

Test case scenarios

These are the test case scenarios that were run:

Baseline definition
This test case uses the four node cluster with Distributed Map mode. The number of workload generating clients were scaled from 25 to 80 to define the base line. See "Baseline definition parameters and results" on page 29.

Vary caching modes
This test case uses the three Trade caching options: No Caching, Command Caching, and Distributed Map caching. For Command Caching, an additional scenario was done with Servlet caching enabled. The objective was to determine which caching mode produced the highest throughput. See "Vary caching mode" on page 31.

Vary garbage collection policy
This test case uses measured the impact of the four possible JVM garbage collection policies that can be set for the JVM. See "Vary garbage collection policy" on page 36.

Vary caching policy
This series of test cases compare the various cache replication sharing policies. These were done for the four sharing types: not-shared, shared-push, shared-pull, shared-push-pull. They were run with the Trade caching option set to Distributed Map. See "Vary caching policy" on page 37.

Vary proxy system in the DMZ
This test case compares the throughput obtained with the WebSphere Edge Services Caching Proxy Server, the WebSphere Proxy Server, and no proxy server.

With no proxy server, the Apache Web server was moved into the DMZ and the Trade workload simulator was configured to direct all request to the Apache Web server. See "Vary proxy system in the DMZ" on page 42.

Scaling the size of the DynaCache
This series of test cases compare the various dynamic cache sizes. They were run with the Trade caching option set to Distributed Map. The dynamic cache size was varied from 2000 to 20000. See "Scaling the size of the DynaCache" on page 45.

Scaling the update part of the Trade workload
Updates from data are the critical case for caching, because the caching mechanism has to make sure that data is consistent system-wide in all caches and variables. In this test case, the update part of the Trade workload was scaled. Normally 14% of the transactions in the Trade workload require a data base update. In this test case, the data base update activity was scaled from 0% to 14% in increments of 2%. See "Scaling the update part of the Trade workload" on page 47.

Enabling DynaCache disk offload
This test case measured the impact of enabling dynamic cache disk offload. See "Enabling DynaCache disk offload" on page 50.

Comparison between IBM System z9, type 2094–S18 and IBM System z10, type 2097–E26
This test case shows the client scaling results from the IBM System z9® compared to the IBM System z10. See "Comparison between IBM System z9, type 2094–S18 and IBM System z10, type 2097–E26" on page 55.

Using z/VM VSWITCH
This test case compares the z/VM network guest configuration when it is changed from guest LAN to VSWITCH. See "Using z/VM VSWITCH" on page 57.

Baseline definition parameters and results

These are the parameters used in the baseline testcase, and the results in terms of throughput and CPU utilization.

This test case uses the four node cluster with Distributed Map Caching mode and the number of workload generating clients were scaled from 25 to 65 to define the base line. The client workload was scaled by increasing the number of users from 25 in increments of 5, until the throughput observed for the cluster peaked.

All runs were done with the standard configuration using the WebSphere Edge Services Caching Proxy Server, described in Table 5.

Table 5. Cache definitions - baseline parameters

DynaCache size	Caching mode	Servlet caching	Replication domain	Replication type
20000	Distribute map	Disabled	Entire domain	Not shared

Figure 15 on page 30 illustrates the throughput results for the baseline tests. Figure 16 on page 31 illustrates the CPU utilization results for the baseline tests.

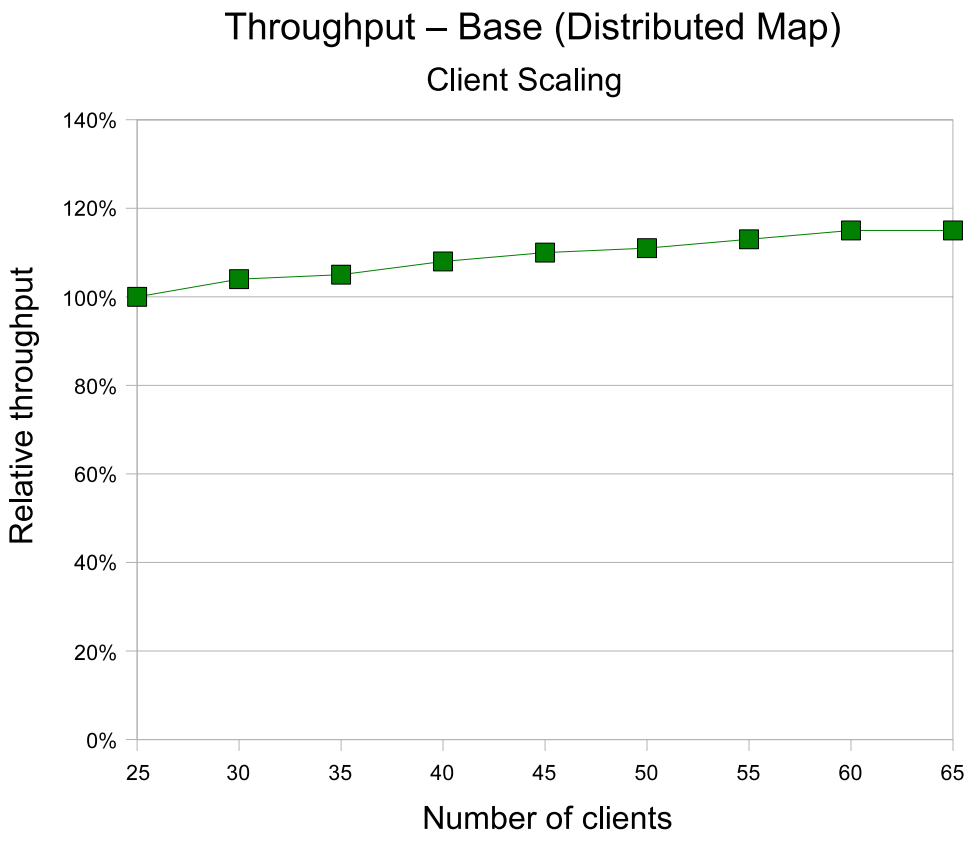

Figure 15. Throughput - Distributed Map Caching, WebSphere Edge Services Caching Proxy

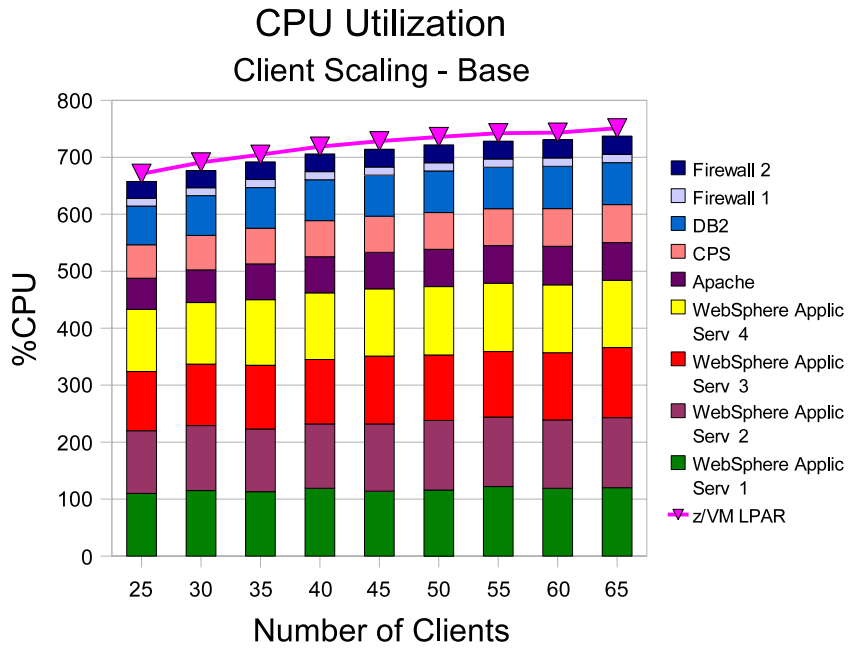

Figure 16. CPU utilization - Distributed Map Caching, WebSphere Edge Services Caching Proxy

Observations

Throughput peaked at 60 workload clients and drops at 65 workload clients. Total CPU utilization is still increasing through 65 workload clients. Total CPU utilization for z/VM is greater than 740% at 55 clients, and increases to 751% at 65 clients. This is approaching CPU saturation.

Conclusions

Increasing the number of workload generating clients increases the throughput until the system is fully utilized.

Vary caching mode

In this test scenario, the Trade application was configured for: No caching, Command caching, and Distributed Map caching. For Command caching, an additional scenario was done with Servlet caching enabled. The objective was to determine which caching mode produced the highest throughput.

The standard configuration with WebSphere Edge Services Caching Proxy Server was used. The client workload was scaled by increasing the number of users from 25 in increments of 5 until the throughput observed for the cluster peaked.

All runs were done with the cache configuration described in Table 6.

Table 6. Cache definitions - vary caching modes

DynaCache size	Caching mode	Servlet caching	Replication domain	Replication type
N/A	No caching	N/A	N/A	N/A

Table 6. Cache definitions - vary caching modes (continued)

DynaCache size	Caching mode	Servlet caching	Replication domain	Replication type
20000	Command caching	Disabled	Entire domain	Not shared
20000	Command caching	Enabled	Entire domain	Not shared
20000	Distributed Map	Disabled	Entire domain	Not shared

Figure 17 illustrates the throughput results for this test scenario. Figure 18 on page 33 illustrates the CPU utilization results for this test scenario.

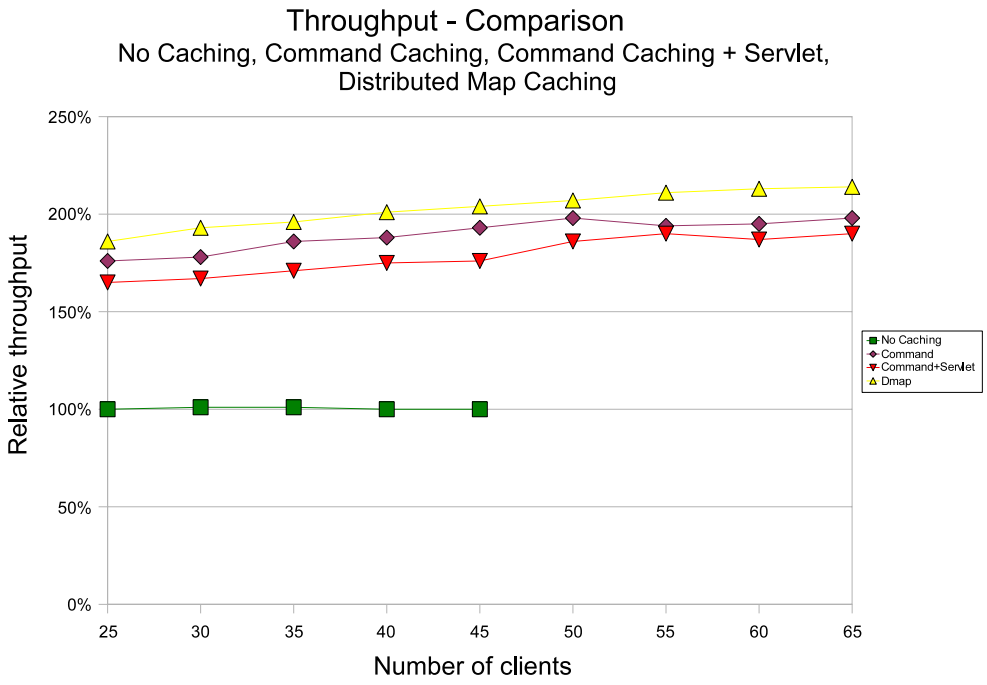

Figure 17. Throughput comparison - No caching, Command caching, Command caching with Servlet caching, Distributed Map caching, WebSphere Edge Services Caching Proxy Server.

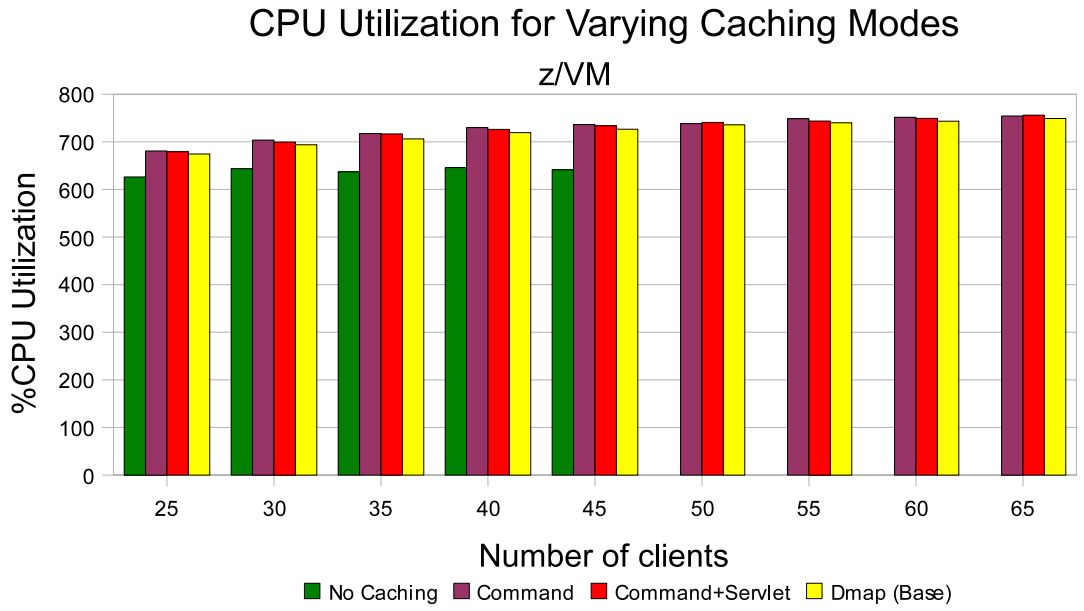

Figure 18. CPU utilization comparison for - No caching, Command caching, Command caching with Servlet caching, Distributed Map caching.

Figure 19 on page 34 illustrates the CPU utilization results expressed relative to the Distributed Map caching results, and taken from the perspective of: the z/VM LPAR, the WebSphere Application Servers, the DB2 server, Apache Web servers, the Caching Proxy Server (CPS), and the firewalls.

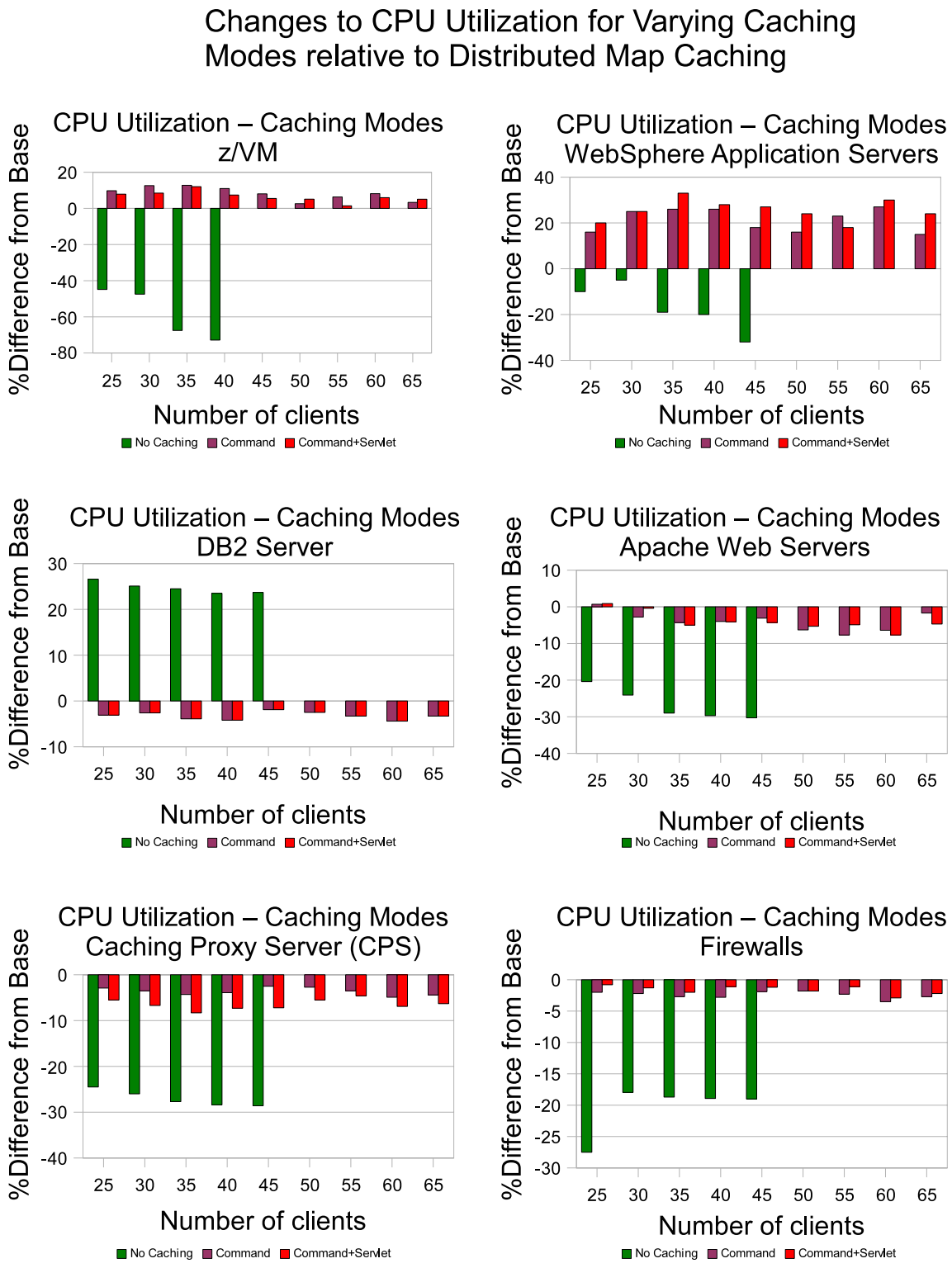

Figure 19. Changes to CPU utilization for No caching, Command caching, Command caching with Servlet caching, and Distributed Map caching, relative to base (Distributed Map caching)

Observations

With no caching being done at the WebSphere Application Servers, the CPU utilization on the database server increases to greater than 95% as more workload clients are added. In the CPU utilization charts above, it can be seen that DB2 utilization is significantly higher for no caching than with the other caching modes. Throughput reaches a maximum at 35 workload clients and starts to drop as more clients are added, while overall CPU utilization remains relatively constant.

When caching is enabled, the throughput is significantly higher in all modes than without caching, and it increases from 25 clients to at 50 or more clients. The CPU load is only a little bit higher.

With Command caching, maximum throughput was reached at 50 workload clients and then decreased or remained the same. Enabling Servlet caching with Command caching results in a throughput peak at 55 workload clients. The maximum throughput obtained is less than what was seen for Command caching without Servlet caching. Adding workload clients above 55 resulted in the throughput tailing off or remaining the same. For both, overall CPU utilization continues to increase as more workload clients are added. Both Command caching and Command caching with Servlet caching have higher WebSphere CPU utilization then Distributed Map caching.

Distributed Map caching achieves the highest throughput at lowest CPU load.

Conclusions

In the no caching case, it appears that throughput performance is being constrained by DB2, and the CPU cost for the transactions is much greater.

The effects of enabling command caching is quite dramatic, DB2 CPU utilization has deceased and throughput has almost doubled. The CPU utilization of the four WebSphere Application Servers in the cluster has also increased, but only by approximately 10% per server.

The Trade file `cachespec.xml` has a servlet section, so Servlet caching with Command caching was enabled. This combination did not result in a throughput improvement. The addition of Servlet caching did not improve the overall throughput; it had a small negative effect on throughput, a decrease of approximately 6%. There is a slight increase in WebSphere Application Server CPU utilization from that observed for Command caching.

Distributed Map caching is handled in application code in the Trade application. An investigation of this code indicated that enabling Servlet caching with Distributed Map caching would not result in any additional improvement, because Servlet caching and Distributed Map caching cache the same objects.

As can be seen in Figure 19 on page 34, the highest throughput was achieved with Trade caching mode set to Distributed Map. Distributed map caching resulted in a throughput increase of more than double what was seen with no caching. It also shows a 6% increase over Command caching, and the overall CPU utilization is the lowest. In particular, the total CPU utilization of the WebSphere Application Servers is lower than all other modes.

Any of the caching options produced a significant performance improvement over no caching. It is recommended that some form of caching be used.

Because Distributed Map caching produced the highest throughput, all subsequent measurements were performed with this option.

Vary garbage collection policy

The impact of the possible JVM garbage collection policies were tested in this scenario.

Table 7 describes the four possible JVM garbage collection policies that can be set for the JVM.

Table 7. Garbage collection policies for JVM

Policy	Option	Description
Optimize for throughput	Xgcpolicy:optthruput (optional)	This is the default policy. It is typically used for applications where overall throughput is more important than short garbage collection pauses. The application is stopped each time that garbage is collected.
Optimize for pause time	Xgcpolicy:optavgpause	Trades off higher throughput for shorter garbage collection pauses by performing some of the garbage collection concurrently. The application is paused for shorter periods.
Generational concurrent	Xgcpolicy:gencon	Handles short-lived objects differently than objects that are long-lived. Applications that have many short-lived objects can see shorter pause times with this policy, while still producing good throughput.
Subpooling	Xgcpolicy:gencon	Uses an algorithm similar to the default policy's, but uses an allocation strategy that is more suitable for multiprocessor machines. This policy is of benefit to SMP machines with 16 or more processors. This policy is available only on IBMSystem p® and IBM System z platforms. Applications that need to scale on large machines can benefit from this policy.

This test case was run to see if changing the JVM's garbage collection policy would have any effect on the throughput. The workload driver was set to 30 clients for all runs and Trade was set to Distributed Map caching. The standard configuration with WebSphere Edge Services Caching Proxy Server was used.

All runs were done with the cache configuration described in Table 8.

Table 8. Cache definitions - Vary garbage collection policy

DynaCache size	Caching mode	Servlet caching	Replication domain	Replication type
20000	Distributed Map	Disabled	Entire domain	Not shared

Figure 20 on page 37 illustrates the throughput using the various garbage collection policies.

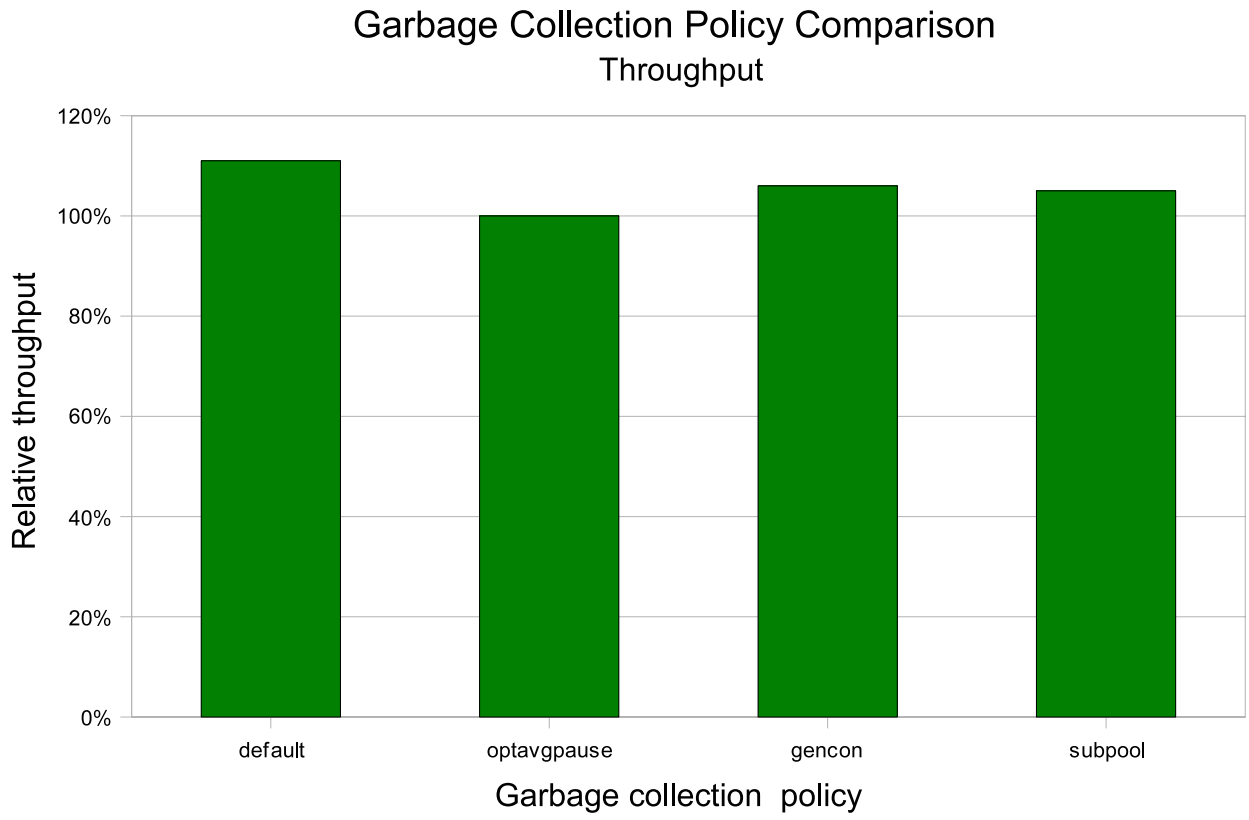

Figure 20. Garbage collection policy comparison

Observations

The default garbage collection policy of optimize for throughput resulted in the highest throughput. The other garbage collection policies resulted in lower throughput with optimize for time having the lowest throughput.

Conclusion

For the Trade workload, the default garbage collection policy produced the best throughput.

Vary caching policy

These are a series of test cases that compare the various cache replication sharing policies.

To run these tests required that the Trade application be modified to include an additional configuration parameter. This parameter was required because the Distributed Map API is used to set the sharing type when objects are inserted into the cache. The original version of Trade always set the type to ***not-shared***.

A new JVM custom property, TRADE_REPLICATION_SHARING_TYPE, was defined. This property is set through the Administration console, **Servers** → **Application Servers** → **server1** → **Process Definition** → **Custom Properties**.

The acceptable values are:

1 Not-shared

2 Shared-push

3 Shared-pull

4 Shared-push-pull

Table 9 explains the various sharing policies.

Table 9. Cache replication sharing policies for JVM

Value	Description
Not-shared	Cache entries for this object are not shared among different application servers. These entries can contain non-serializable data. For example, a cached servlet can place non-serializable objects into the request attributes, if the <class> type supports it.
Shared-push	Cache entries for this object are automatically distributed to the DynaCaches in other application servers or cooperating Java virtual machines (JVMs). Each cache has a copy of the entry at the time it is created. These entries cannot store non-serializable data.
Shared-pull (Deprecated)	Cache entries for this object are shared between application servers on demand. If an application server gets a cache miss for this object, it queries the cooperating application servers to see if they have the object. If no application server has a cached copy of the object, the original application server issues the request and generates the object. These entries cannot store non-serializable data. This mode of sharing is not recommended.
Shared-push-pull	Cache entries for this object are shared between application servers on demand. When an application server generates a cache entry, it broadcasts the cache ID of the created entry to all cooperating application servers. Each server then knows whether an entry exists for any given cache ID. On a given request for that entry, the application server knows whether to generate the entry or retrieve it from somewhere else. These entries cannot store non-serializable data.

These tests were run with a number of workload clients starting at 25 and increased by 5 until a total of 60 clients was reached.

All runs were done with the cache configuration described in Table 10.

Table 10. Cache definitions - Vary caching policy

DynaCache size	Caching mode	Servlet caching	Replication domain	Replication type
20000	Distributed map	Disabled	Entire domain	Not-shared
20000	Distributed map	Disabled	Entire domain	Shared-push
20000	Distributed map	Disabled	Entire domain	Shared-pull
20000	Distributed map	Disabled	Entire domain	Shared-push-pull

Figure 21 on page 39 illustrates the throughput results with various caching policies. Figure 22 on page 40 illustrates the CPU utilization results with various

caching policies.

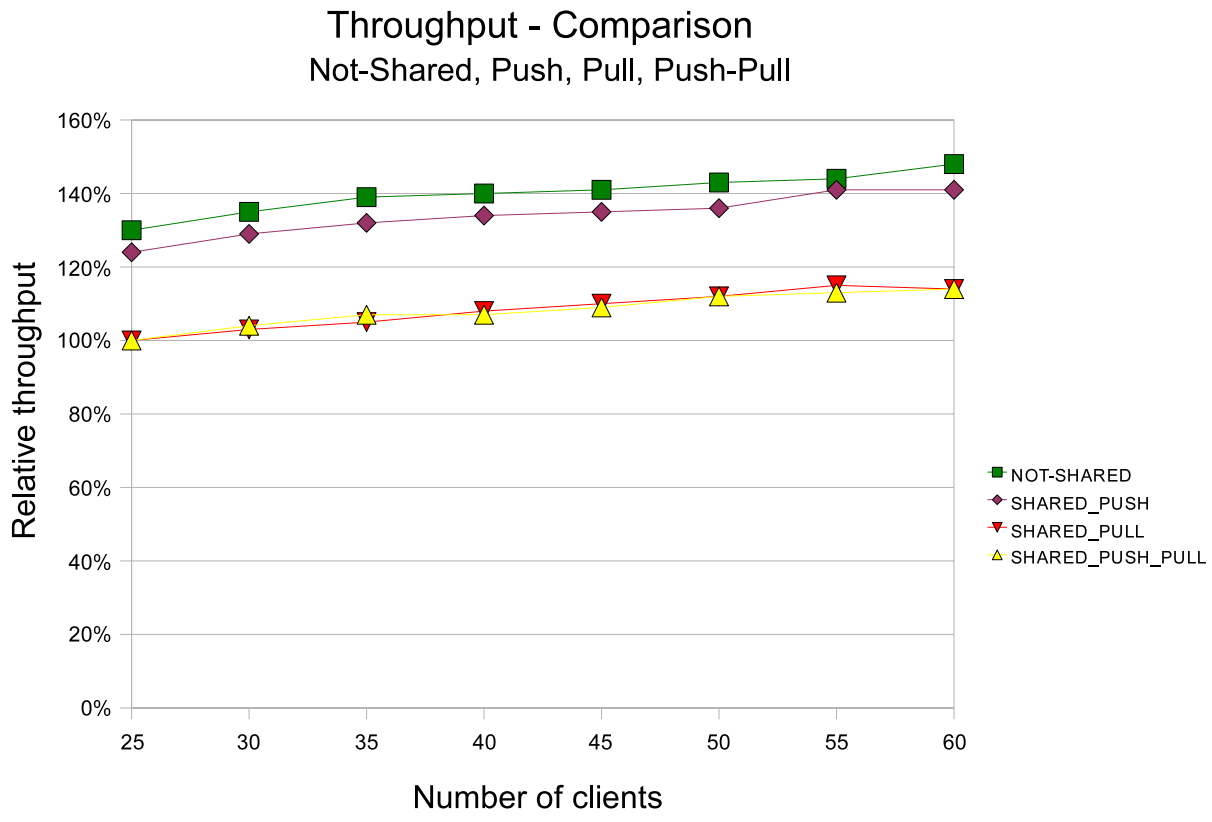

Figure 21. Throughput comparison with caching policy not-shared, shared-push, shared-pull, shared-push-pull

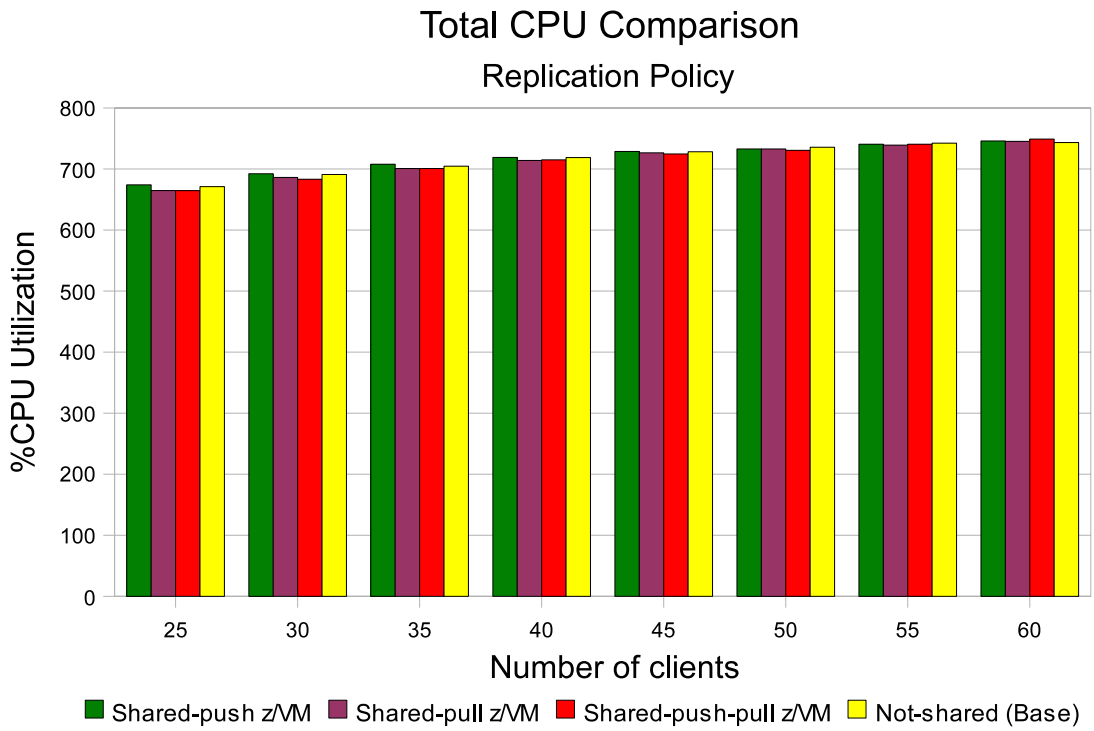

Figure 22. CPU utilization comparison for caching policy not-shared, shared-push, shared-pull, and shared-push-pull

Figure 23 on page 41 illustrates the CPU utilization results expressed relative to the not shared caching results, and taken from the perspective of: the z/VM LPAR, the WebSphere Application Server, the DB2 server, Apache Web servers, the Caching Proxy Server (CPS), and the firewalls.

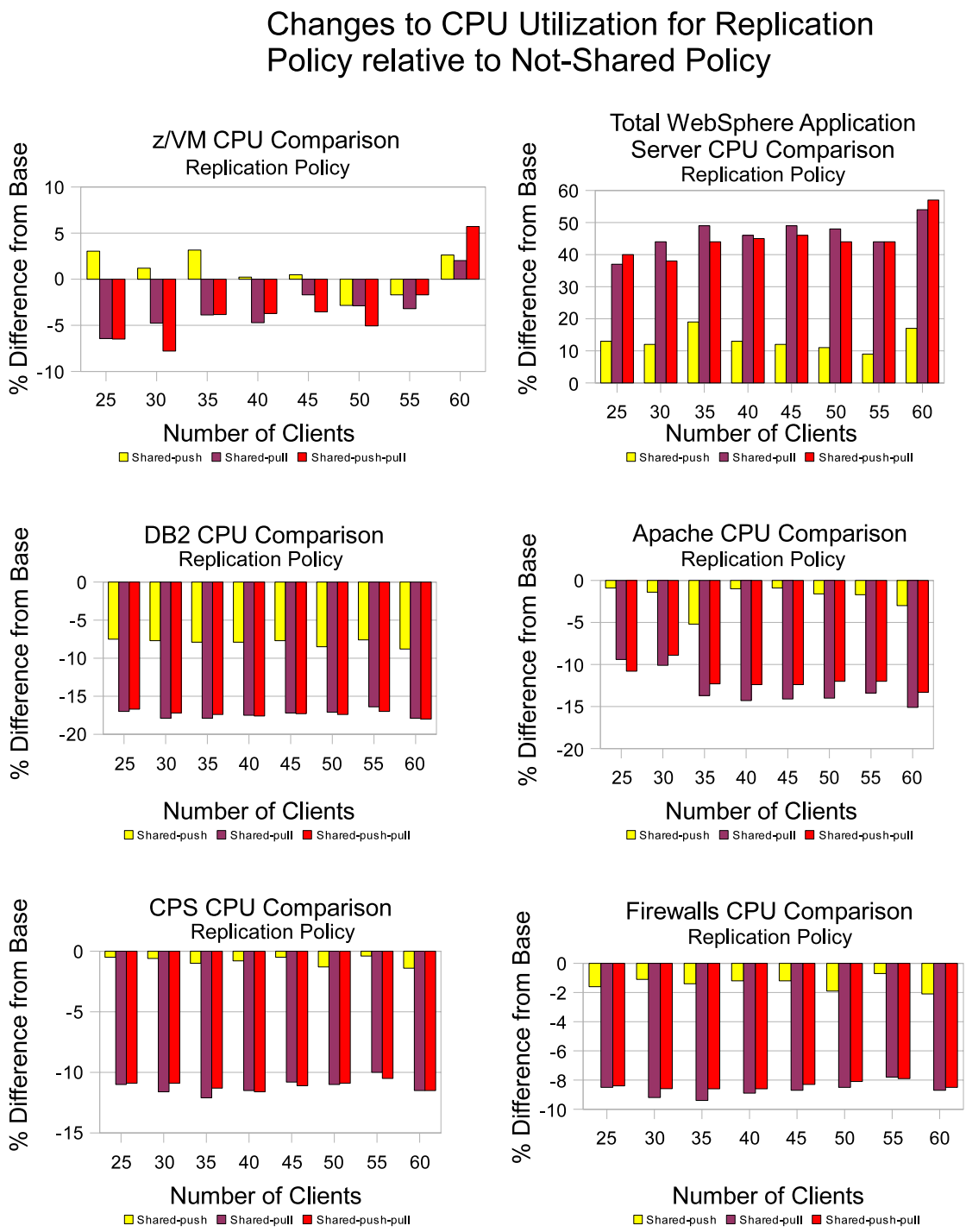

Figure 23. Changes in CPU utilization for Caching policy shared-push, shared-pull, shared-push-pull relative to base (caching policy not-shared)

Observations

In the three cases, shared-push, shared-pull and shared-push-pull throughput peaked at 55 workload clients. Overall CPU utilization continued to increase through 60 workload clients. The highest throughput was achieved with the not-shared case, the second highest was reached with the shared-push policy, the shared-pull and shared-push-pull cases had the lowest throughput.

The total CPU utilization for the four cases is very similar, varying by less than 3 %. Looking at each individual component shows that there is a significant increase in WebSphere CPU utilization for the shared-pull and shared-push-pull cases and corresponding decrease in CPU utilization on the other systems.

Conclusion

The not-shared cache replication policy produced the highest throughput. The throughput with shared-push's is lower than the not-shared case by approximately 5%. The throughput for shared-pull and shared-push-pull are both significantly lower while, the WebSphere CPU utilization has increased significantly. This indicates that for this workload, the overhead related with the share-pull and shared-push-pull cache replication mode is significant. However, even shared-pull does not improve the performance. There is more WebSphere cache replication activity with a corresponding decrease in throughput.

In this environment, using the not-shared policy produced the best results. It is more expensive to fetch from a peer cache than to obtain data from the database, because four caches have to be synchronized.

Vary proxy system in the DMZ

This test case compared the results of the tests run with three different environments for the proxy system.

This test case compared the results of the following three environments, to show the impact of each scenario:

- Trade transactions sent to the WebSphere Edge Services Caching Proxy Server (CPS)
- Trade transactions sent to the WebSphere Proxy Server (PS)
- Trade transactions sent directly to the Apache web server (no proxy)

With no proxy server, it was necessary to move the Apache Web server into the DMZ to route the requests forward into the internal zone, and the Trade workload simulator is configured to direct all requests to the Apache Web server.

Table 11 shows how the guest setup was varied for this test case.

Table 11. Guest configuration for varying the proxy system in the DMZ

	CPS		PS		No Proxy	
System/Guest	**Memory size**	**Number of CPUs**	**Memory size**	**Number of CPUs**	**Memory size**	**Number of CPUs**
z/VM	1248 MB	8	1248 MB	8	1248 MB	8
Caching Proxy Server (CPS)	512 MB	1	–	–	–	–
WebSphere Proxy Server (PS)	–	–	750 MB	1	–	–
Apache HTTP Server	512 MB	1	–	–	512 MB	1
WebSphere Application Server 1	2 GB	2	2 GB	2	2 GB	2
WebSphere Application Server 2	2 GB	2	2 GB	2	2 GB	2
WebSphere Application Server 3	2 GB	2	2 GB	2	2 GB	2
WebSphere Application Server 4	2 GB	2	2 GB	2	2 GB	2
DB2 Server	4 GB	1	4 GB	1	4 GB	1
Firewall 1	512 MB	1	512 MB	1	512 MB	1

Table 11. Guest configuration for varying the proxy system in the DMZ (continued)

	CPS		PS		No Proxy	
Firewall 2	512 MB	1	512 MB	1	512 MB	1

All runs were done with the cache configuration described in Table 12.

Table 12. Cache definitions - vary proxy system in the DMZ

DynaCache size	Caching mode	Servlet caching	Replication domain	Replication type
20000	Distribute map	Disabled	Entire domain	Not shared

Figure 24 illustrates the throughput comparison for three different proxy systems.

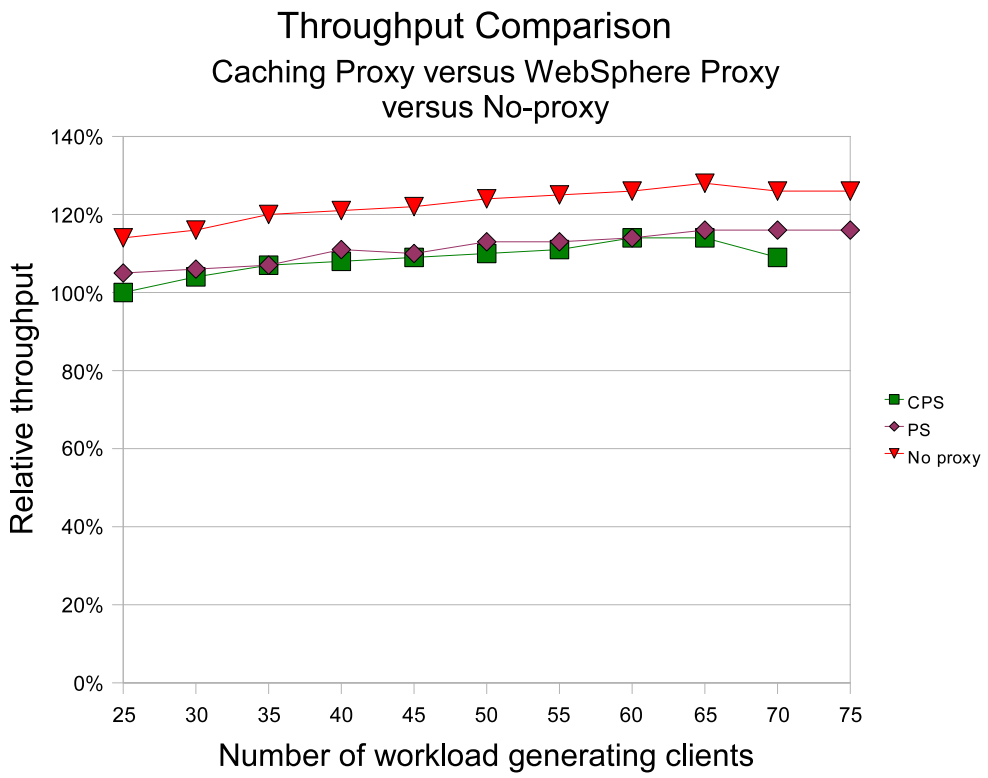

Figure 24. Throughput comparison - CPS , PS, and no proxy

Figure 25 on page 44 shows the CPU load per WebSphere node to demonstrate how the load balancing works for the various scenarios.

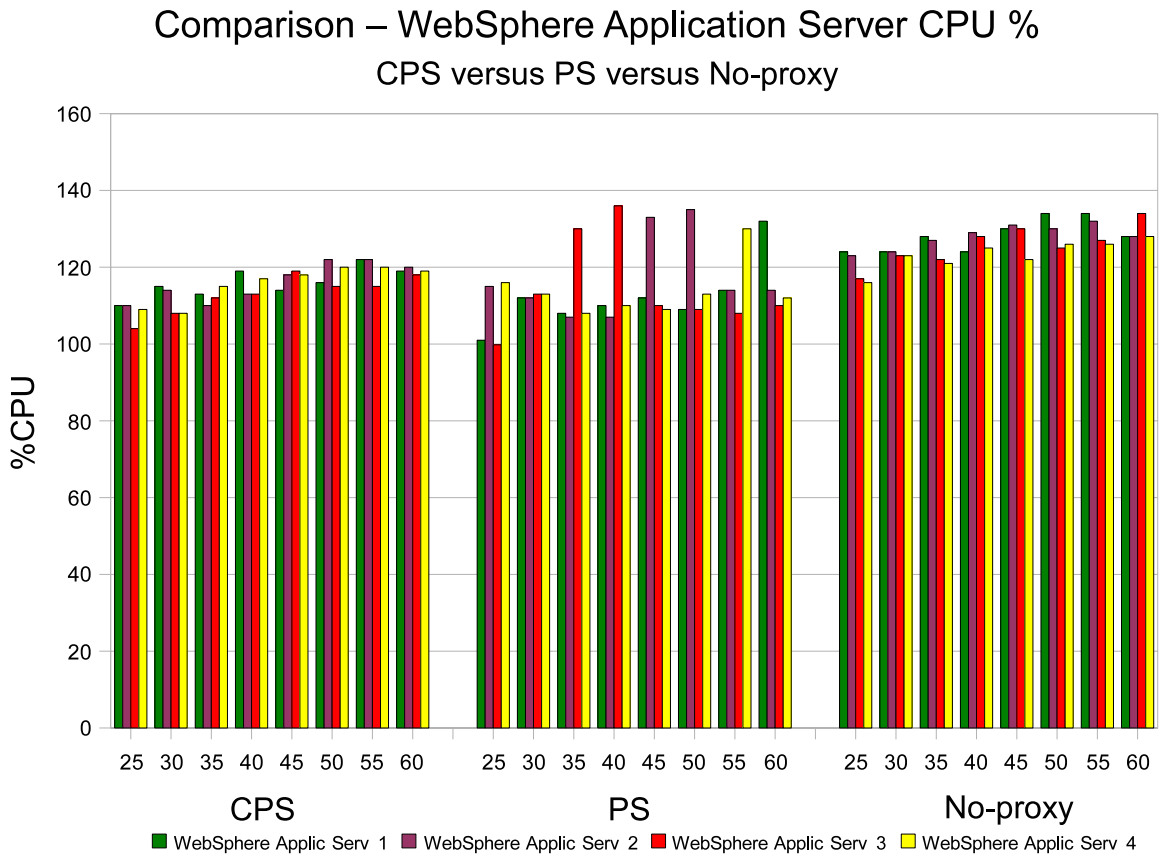

Figure 25. CPU utilization of the WebSphere nodes when varying the proxy.

Observations

Replacing the WebSphere Edge Services Caching Proxy Server with the WebSphere Proxy Server resulted in a slight increase in throughput.

Bypassing a proxy server produced the highest throughput. In the two cases (bypassing the proxy server and the WebSphere Edge Services Caching Proxy Server), the workload balancing function (distributing transaction to the cluster members) was handled in the WebSphere Apache plugin.

In the WebSphere Proxy Server case, the workload balancing function is performed on the proxy server. In all cases, the workload was balanced using a round robin algorithm with equal weighting to each cluster member.

In Figure 25, the WebSphere Proxy Server load balancing function appears to be scheduling the transaction unevenly. However, it does not appear to have an effect on overall throughput.

Conclusions

The best performance was achieved when a proxy server was omitted in the configuration. While this might produce more throughput, it has significant security deficiencies. The proxy functionality is missing (which interprets the TCP/IP packets) and it should be avoided to expose a functional component such

as the Web server to direct access from an insecure zone. Therefore, this is not recommended for production environments. There is only a slight difference between both proxy solutions.

Note: Be aware that removing the proxy means that an important security function is removed, which the Web server can not replace. Also, moving a functional component (Web server) into the DMZ opens the system for attacks. This was done just for testing purpose and is not recommended for production systems.

All subsequent tests were run using the WebSphere Proxy Server, because it is the strategic solution from WebSphere. In that environment, the Apache Web server is no longer used.

The CPU load for the new base is shown in Figure 26.

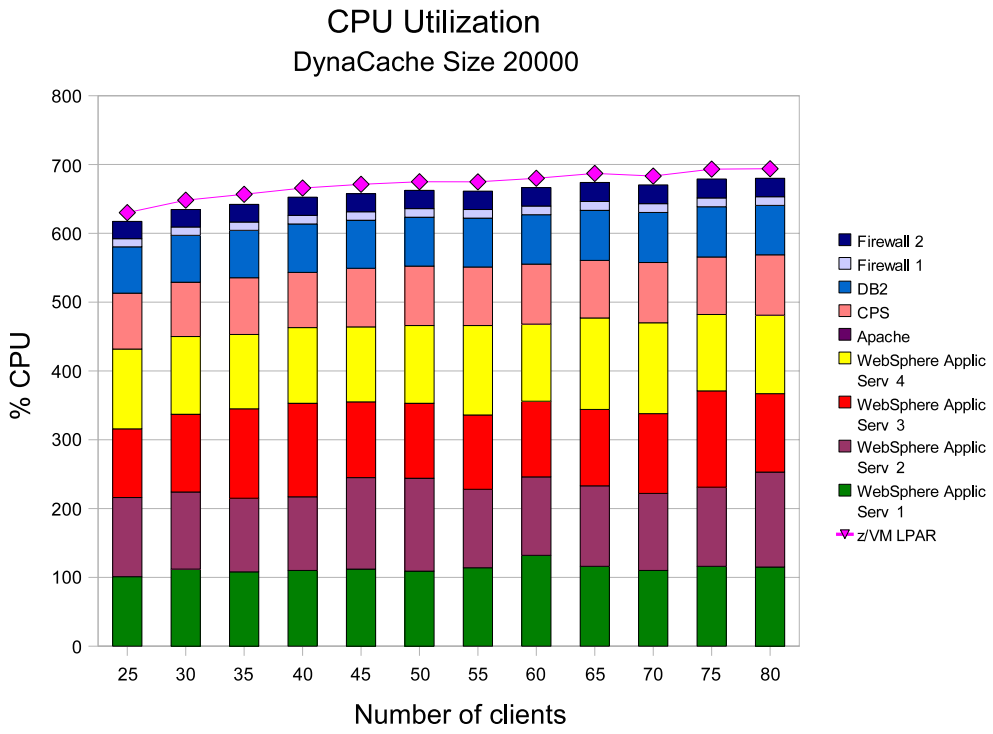

Figure 26. CPU utilization base, Distribute Map Caching, DynaCache size=20000, WebSphere Proxy Server

Scaling the size of the DynaCache

This is a series of test cases that compare the various dynamic cache sizes.

These tests were run with the Trade caching option set to Distributed Map. The dynamic cache size was varied from 2000 to 20000 statements. The values used were 2000, 5000, 10000, 15000, and 20000. These runs were done using the WebSphere Proxy Server and bypassing the Apache Web server.

These tests were run with workload clients starting at 25 and increasing by 5 until a total of 60 clients.

All runs were done with the cache configuration described in Table 13.

Table 13. Cache definitions - Scaling the size of the DynaCache

DynaCache size	Caching mode	Servlet caching	Replication domain	Replication type
2000, 5000, 10000, 20000	Distributed map	Disabled	Entire domain	Not shared

Figure 27 illustrates the throughput when using various dynamic cache sizes.

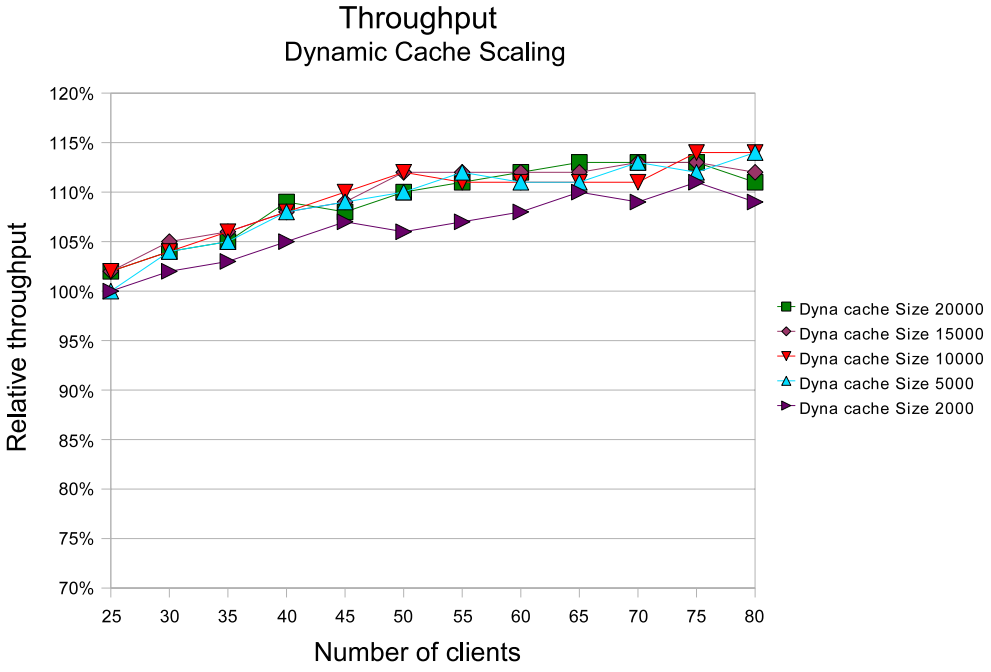

Figure 27. Throughput - Dynamic Cache Size Scaling, WebSphere Proxy Server

Observations

A dynamic cache size of 2000 (the default) had the lowest throughput. Dynamic cache sizes, 5000, 10000, 20000 are very close, where the throughput with a cache size of 5000 statements is mostly the lowest from these.

Conclusions

The default cache size is not the best for this environment. Any cache size above 10000 seems to be suitable for this environment.

Figure 28 on page 47 illustrates the CPU utilization when using various dynamic cache sizes.

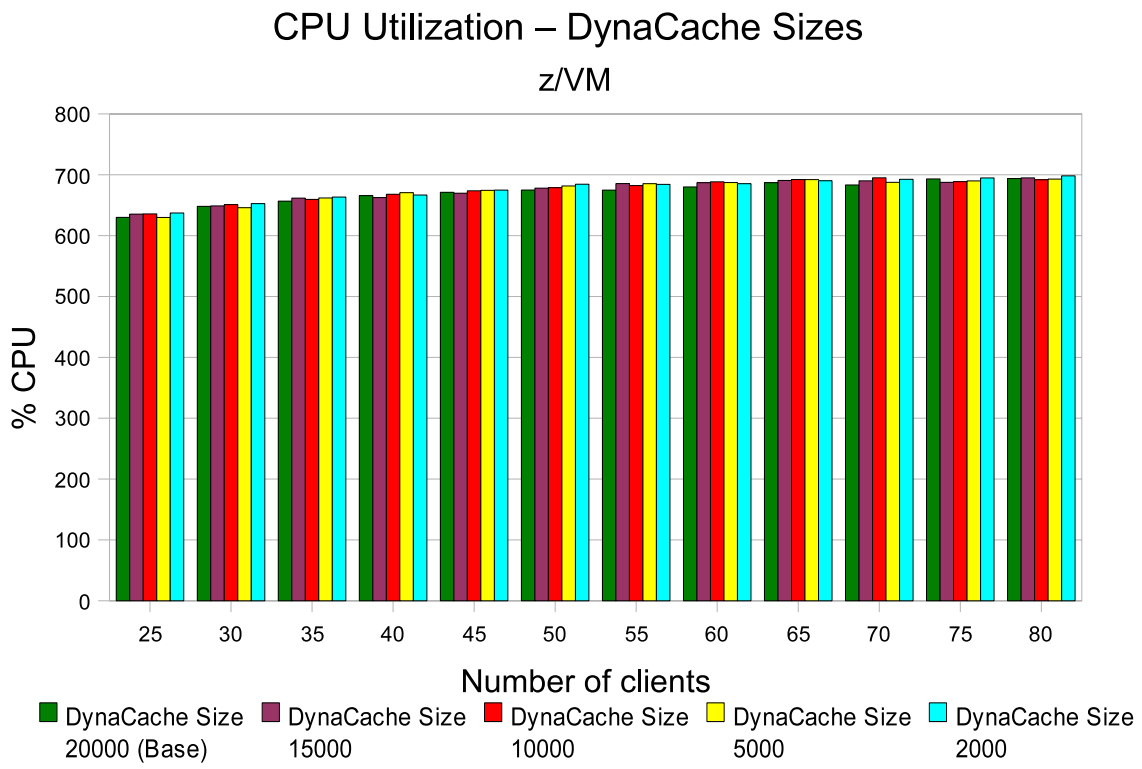

Figure 28. CPU utilization - DynaCache sizes z/VM

Observations

The total CPU utilization results for the various DynaCache sizes are all within 2 % of each other. The DynaCache size of 2000 has a higher WebSphere CPU utilization and corresponding lower throughput. This would seem logical because there is a smaller amount of memory available to hold cached data and would require more WebSphere processing to satisfy the cache misses.

Conclusion

Reducing DynaCache sizes from 20,000 statements to smaller values for this environment did not produce any significant change in either CPU utilization or throughput.

Scaling the update part of the Trade workload

In this test case, the update part of the Trade workload was scaled.

Updates from data are the critical case for caching, because the caching mechanism has to make sure that data is consistent system wide in all caches and variables. Normally 14% of the transactions in the Trade workload require a data base update.

The Trade transactions in the workload are analyzed in Table 14.

Table 14. Trade transactions in the workload

Transaction type	Percent	Requires DB update
Update account	4	Yes

Table 14. Trade transactions in the workload (continued)

Transaction type	Percent	Requires DB update
Buy or sell stock	8	Yes
Register new user	2	Yes
Total for updates	**14**	
Home page	20	No
Account	10	No
Portfolio	12	No
Quotes	40	No
Logoff	4	No
Total for read only	**86**	

In this test case, the data base update activity was scaled from 0% to 14% in increments of 2%. The number of workload clients was set to 65 for all tests and the percent of update transactions was varied according to the data in Table 15.

Table 15. Percent of update transaction scaling

Transaction type	Test case transaction update %							
Update account	0	0	2	4	4	4	4	4
Buy or sell	0	0	0	0	2	4	6	8
Register new user	0	2	2	2	2	2	2	2
Total % of updates	0	2	4	6	8	10	12	14

The environment with the WebSphere Proxy Server (PS) was used for these tests. The initial test, with the number of virtual CPUs (vCPUs) set to one for the Proxy Server, indicated that the Proxy Server CPU utilization was a bottleneck. Additional tests were performed with the number of virtual CPUs set to two from the Proxy Server (without changing the amount of physical CPUs).

All runs were done with the cache configuration described in Table 16.

Table 16. Cache definitions - Scaling the update part of the Trade workload

DynaCache size	Caching mode	Servlet caching	Replication domain	Replication type
10000	Distributed Map	Disabled	Entire domain	Not shared

Figure 29 on page 49 illustrates the throughput when scaling the update part of the Trade workload. Figure 30 on page 49 illustrates the CPU utilization when scaling the update part of the Trade workload.

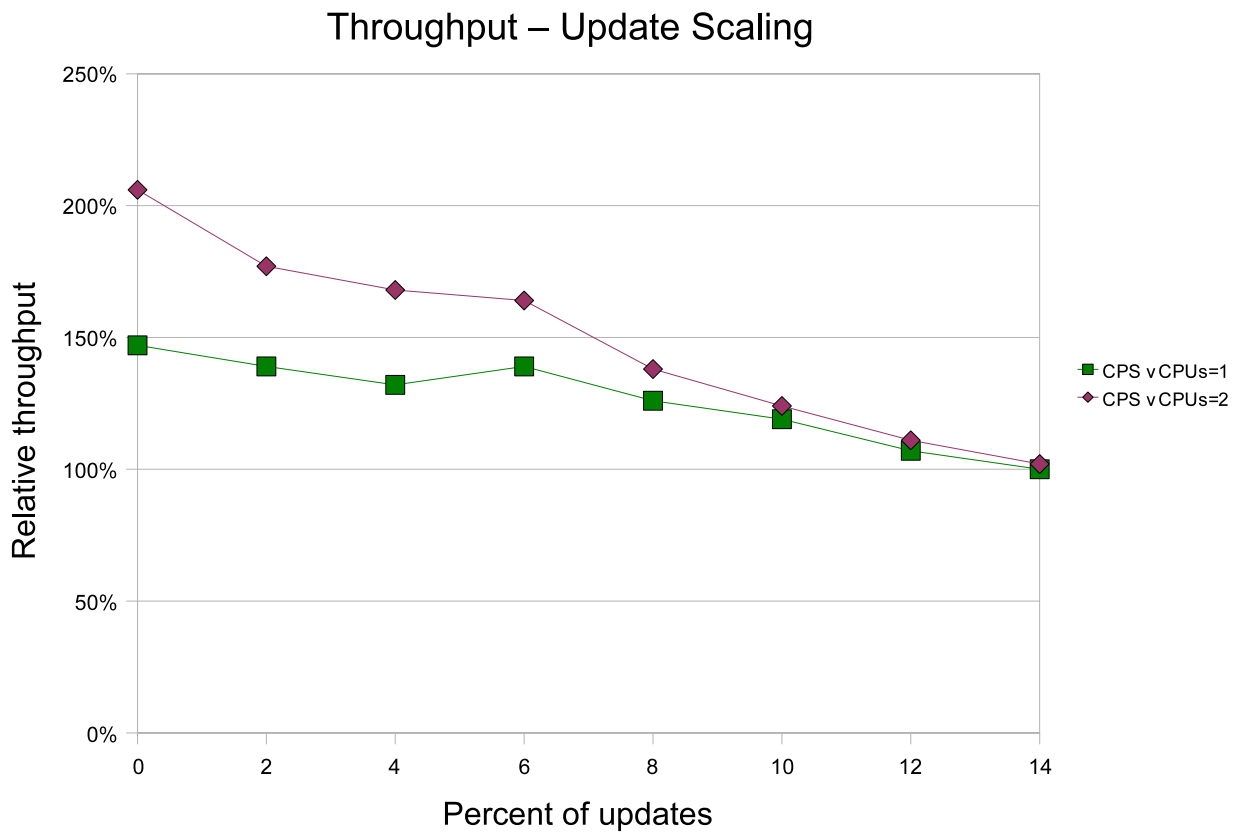

Figure 29. Throughput - Percent of update transaction varied, WebSphere proxy server CPUs set to one and two

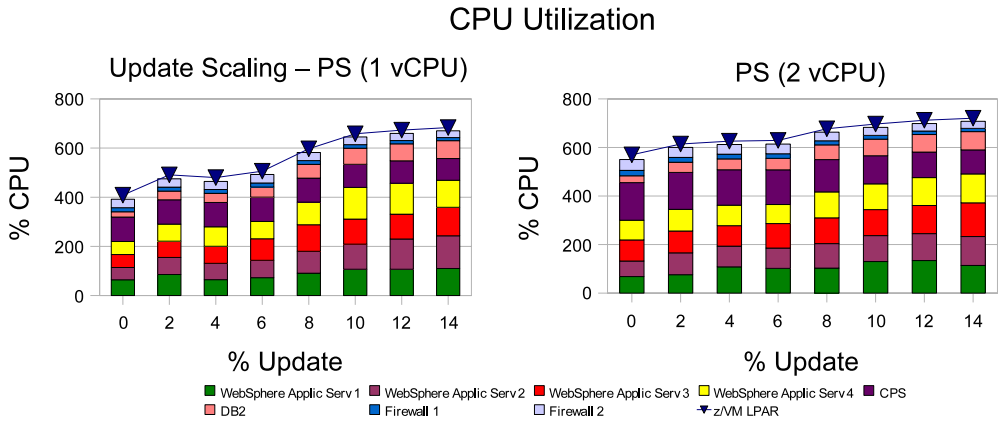

Figure 30. CPU utilization - Trade Workload scaling of update transactions.

Observations

As the percentage of update transactions in the Trade workload is increased from 0% to 14%, the throughput decreases, as expected. Increasing the number of vCPUs for the Proxy Server guest produced a significant throughput improvement until the 8% update case was reached.

As the percent of update transaction increases, the overall CPU utilization increases to its maximum observed value at the 14% case.

Conclusion

For this workload environment, enabling WebSphere DynaChache results in the highest performance as the amount of update transactions are decreased, as expected.

If a workload transaction mix has a very small percentage of update transactions, enabling dynamic cache should produce a significant performance advantage because of the higher cache hit ratio.

Enabling DynaCache disk offload

In these tests, the dynamic cache size was adjusted to a value that would allow the cache to offload entries from real memory to disk.

The number of workload generating clients was set to 65 and the disk offload was enabled. The eviction policy was set to random. The dynamic cache size had to be reduced to less than 2000 to obtain any observable data off-loaded. The smallest value that WebSphere would accept for a Dynamic Cache size was 100 statements.

The environment with the WebSphere Proxy Server was used for this test. In addition to the normal workload, the modified Trade workload simulator script from the previous run was used, which had removed all transactions that resulted in a data base update for one of the runs (0% update part).

All runs were done with the cache configuration described in Table 17.

Table 17. Cache definitions - Enabling DynaCache disk offload

DynaCache size	Caching mode	Servlet caching	Replication domain	Replication type
2000, 1000, 500, 100	Distributed map	Disabled	Entire domain	Not-shared

Figure 31 on page 51 illustrates the throughput when adjusting the dynamic cache size to permit offloading of entries from memory to disk, both with and without updates. Figure 32 on page 51 illustrates the CPU utilization when adjusting the dynamic cache size to permit offloading of entries from memory to disk, both with and without updates.

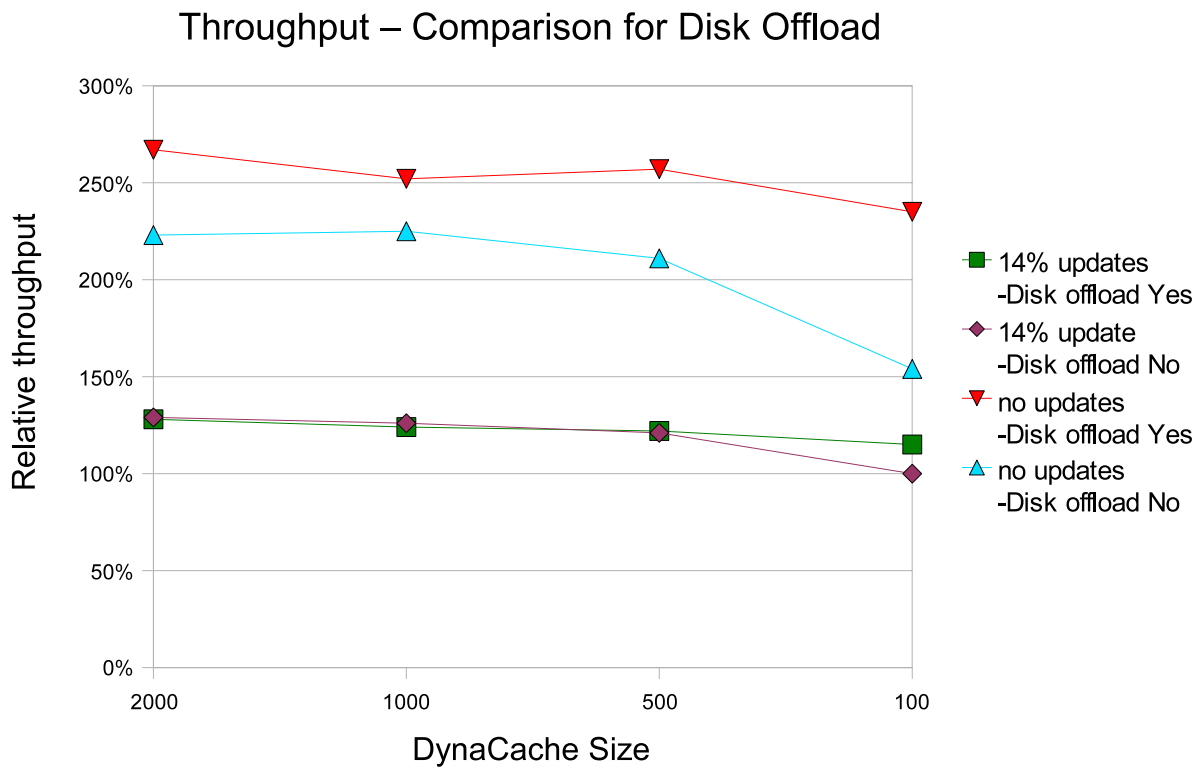

Figure 31. Throughput - Comparison disk offload versus no disk offload with and without updates.

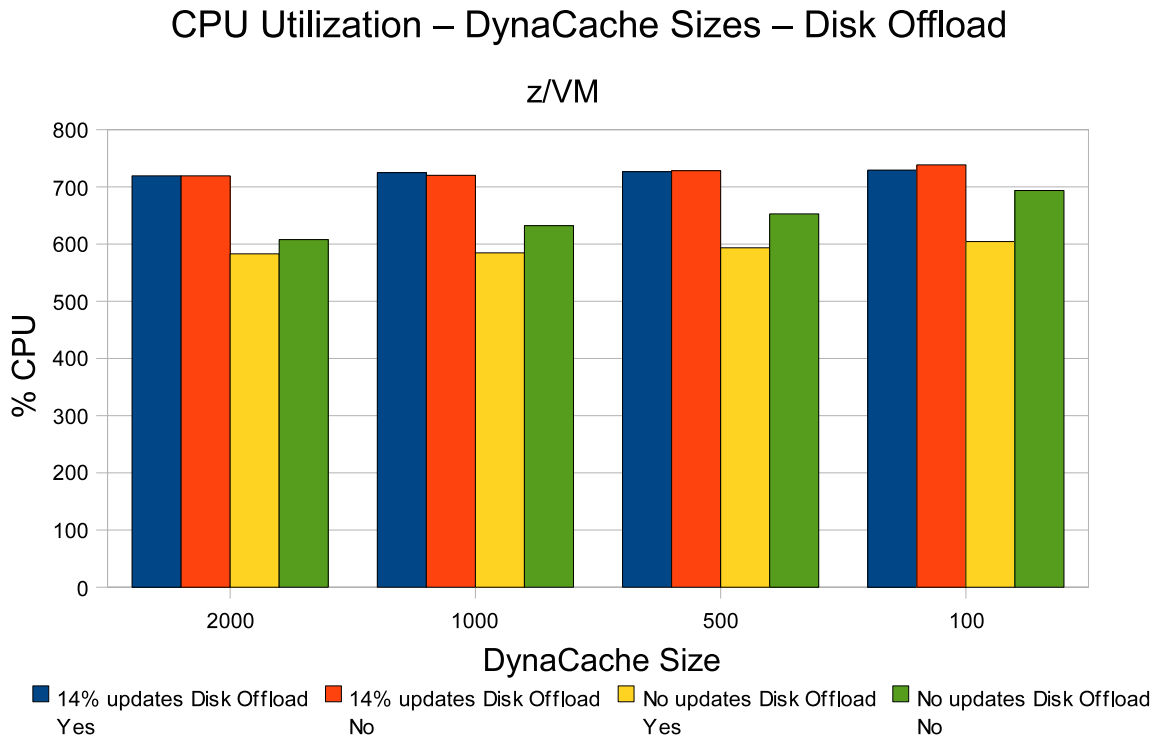

Figure 32. CPU utilization comparison - disk offload versus no disk offload with and without updates.

Figure 33 on page 53 illustrates the CPU utilization results when using disk offload, expressed relative to the results using a cache size of 20,000 and not using disk offload, and taken from the perspective of: the z/VM LPAR, the WebSphere Application Server, the DB2 server, the disk offload proxy server, and the firewalls.

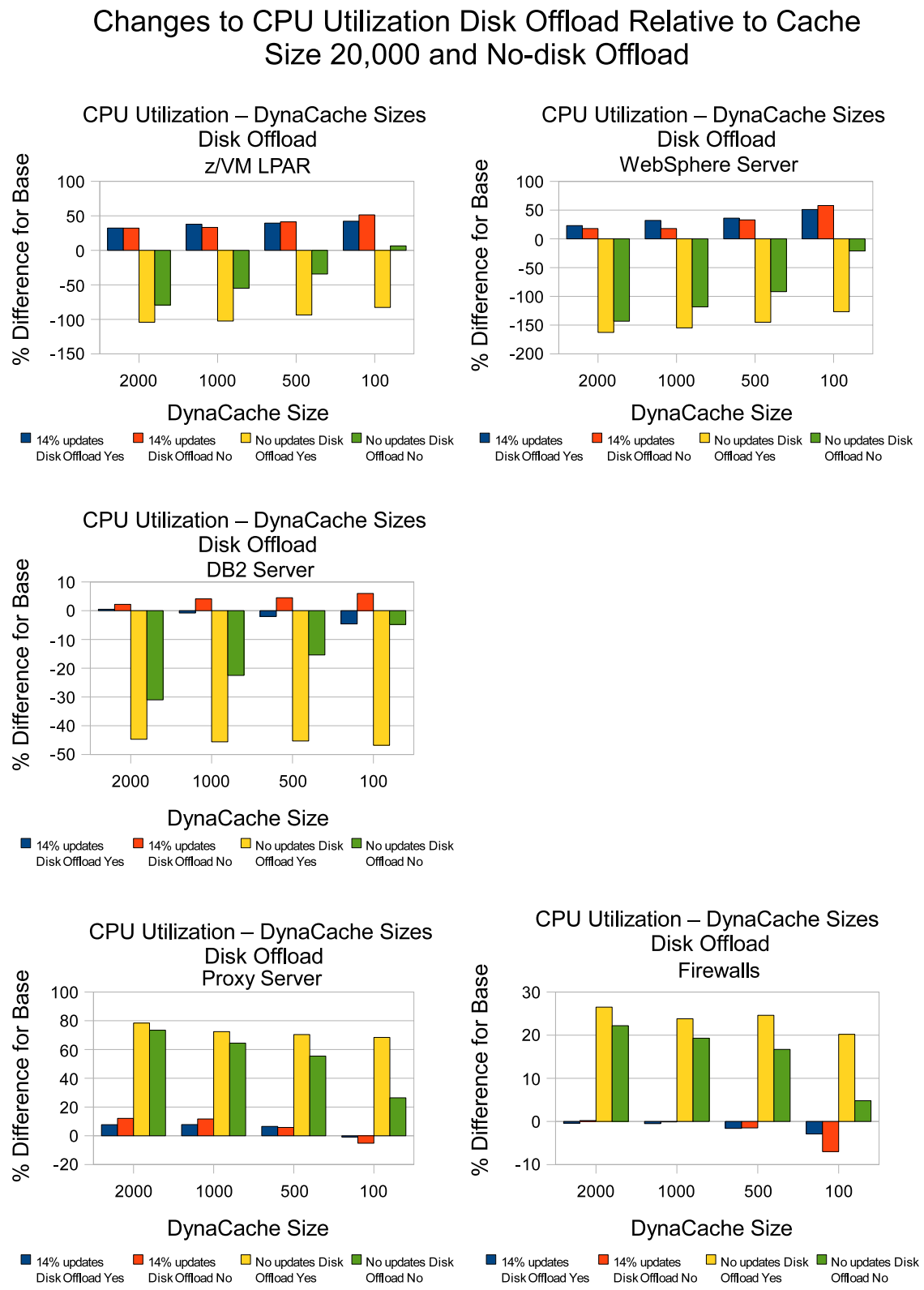

Figure 33. Changes in CPU utilization for DynaCache Disk Offloading relative to base (cache size 20,000 and no-disk offload).

Observations

For the 14% update (normal), Trade workload disk offload produced no improvement in throughput until the dynamic cache size was set to its smallest value (100). At this value, disk offloading increased throughput by 15%. CPU utilization for these cases was almost identical.

It was noticed that the Proxy Server CPU utilization was greater than 90% for the runs with the 14% update Trade workload. For the no update Trade workload, the Proxy Server CPU utilization increases further, according to the increased throughput, and reaches the full CPU utilization. That means, it was gating the throughput. Therefore, the number of virtual CPUs assigned to the Proxy Server guest was increased to two for the no update Trade workload tests.

For the no update Trade workload, disk offloading resulted in an immediate advantage and the advantage increases as the dynamic cache size decreases.

Figure 33 on page 53 shows the advantage of increasing the number of CPUs.

An interesting observation is the relationship between the no update Trade workload case in the disk offloading tests, and the results seen in DynaCache scaling section, Figure 29 on page 49. At a dynamic cache size of 100 the disk offload throughput case is only 11% less than the maximum throughput seen in the no update Trade workload update case, seen above, with a dynamic cache size of 10000.

At a dynamic cache size of 100, disk offloading improved throughput by more than 50%.

In total, there is a slight reduction of CPU utilization for the z/VM LPAR, especially in the no update case.

The CPU utilization for the proxy server and firewall increases over the base results for the no update case. This increase is related to the increased throughput obtained in the no update case.

It is interesting to notice the difference in the CPU utilization for WebSphere and DB2 for the no update case. With disk offloading set to yes and a DynaCache size of 100, there is a significant reduction in CPU utilization versus the disk offloading set to no case.

Conclusion

Disk offload can provide a significant performance improvement for the right application transaction mix and can reduce the CPU load. Even in the case where it provides no significant improvement, it provides no additional overhead. That means that in the tested environment, this feature could be used to run with a significantly smaller cache size, for example of 2000 statements, to save memory without performance degradation.

Note: Disk offloading is a very important DynaCache feature.

Comparison between IBM System z9, type 2094–S18 and IBM System z10, type 2097–E26

This test compares the results for client scaling for runs done on the IBM System z9, with the results on an IBM System z10.

This cache setup is described in Table 18.

Table 18. Cache definitions - IBM System z9 versus IBM System z10 test

DynaCache size	Caching mode	Servlet caching	Replication domain	Replication type
20000	Distributed map	Disabled	Entire domain	Not-shared

The test used one LPAR on a 16-way IBM System z10, equipped with same amount of physical resources as the IBM System z9 LPAR. The guest definitions were the same, except that in the IBM System z10 environment the number of virtual CPUs assigned to the Proxy Server guest was increased from one to two because the single CPU was fully utilized and gating the throughput. Table 19 described the test configuration.

Table 19. Guest configuration for IBM System z10 environment

System/Guest	Memory size	Number of CPUs
z/VM	1248 MB	8
Caching Proxy Server (CPS)	-	-
WebSphere Proxy Server (PS)	750 MB	1/2
WebSphere Application Server 1	2 GB	2
WebSphere Application Server 2	2 GB	2
WebSphere Application Server 3	2 GB	2
WebSphere Application Server 4	2 GB	2
DB2 Server	4 GB	1
Firewall 1	512 MB	1
Firewall 2	512 MB	1

Figure 34 on page 56 compares the throughput for the two systems when the number of workload clients is scaled from 40 to 70.

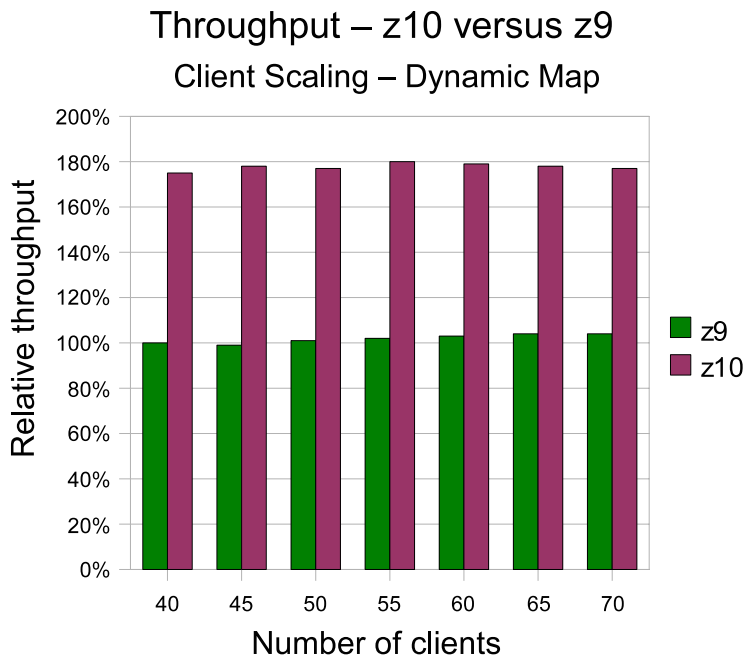

Figure 34. Throughput comparison - client scaling, z9 versus z10, WebSphere Proxy Server.

Figure 35 compares the CPU utilization for the two systems when the number of workload clients is scaled from 40 to 70.

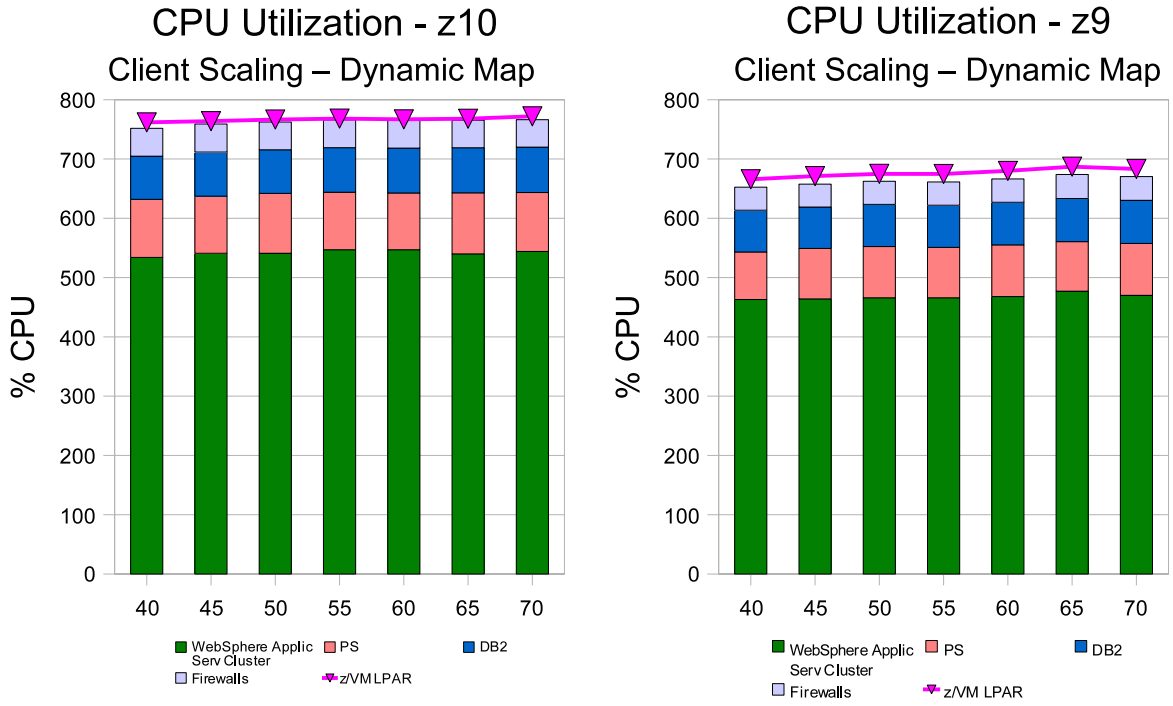

Figure 35. CPU utilization comparison - client scaling, z9 versus z10, WebSphere Proxy Server.

Observations

Replacing the IBM System z9 system with an IBM System z10 system resulted in a throughput increase of 86%. WebSphere cluster utilization increased by 18%. The WebSphere Proxy Server utilization increased by 28%, and the DB2 and Firewall utilization showed a slight increase. Total z/VM utilization is peaking at 772%.

Conclusions

The IBM System z10 system obtained a significant throughput advantage over the IBM System z9 system. The Trade workload used in this test demonstrates the performance advantage of the IBMSystem z10 system.

Using z/VM VSWITCH

In this test case, the network interface for interconnecting the z/VM guest systems was changed from a HiperSockets configured guest LAN to a VSWITCH configured LAN.

The following z/VM commands were used to set up the two VSWITCH network interfaces.

```
DEF VSWITCH testsw1 RDEV none CONNECT VLAN 1 PORTTYPE ACCESS CONTROLLER DTCVSW1
PRIROUTER PORTNAME PORT30

SET VSWITCH testsw1 GRANT DTCVSW1 vlan 1
SET VSWITCH testsw1 GRANT LNX00085 vlan 1
SET VSWITCH testsw1 GRANT LNX00090 vlan 1
SET VSWITCH testsw1 GRANT LNX00091 vlan 1
DEF VSWITCH testsw2 RDEV none CONNECT VLAN 2 PORTTYPE ACCESS CONTROLLER DTCVSW1
PRIROUTER PORTNAME PORT30

SET VSWITCH testsw2 GRANT DTCVSW1 vlan 2
SET VSWITCH testsw2 GRANT LNX00080 vlan 2
SET VSWITCH testsw2 GRANT LNX00081 vlan 2
SET VSWITCH testsw2 GRANT LNX00082 vlan 2
SET VSWITCH testsw2 GRANT LNX00083 vlan 2
SET VSWITCH testsw2 GRANT LNX00084 vlan 2
SET VSWITCH testsw2 GRANT LNX00086 vlan 2
SET VSWITCH testsw2 GRANT LNX00090 vlan 2
```

To connect a guest to the VSWITCH, run the following two CP commands to define a nic and then couple the nic to the VSWITCH.

For VSWITCH testsw1:

```
DEFINE NIC 6200 QDIO
COUPLE 6200 TO SYSTEM testsw1
```

For VSWITCH testsw2:

```
DEFINE NIC 6203 QDIO
COUPLE 6203 TO SYSTEM testsw2
```

The environment with the WebSphere proxy server was used for this test.

All runs were done with the cache configuration described in Table 20.

Table 20. Cache definitions - Using z/VM VSWITCH

DynaCache size	Caching mode	Servlet caching	Replication domain	Replication type
20000	Distributed map	Disabled	Entire domain	Not shared

Figure 36 illustrates the throughput when using a guest LAN versus a VSWITCH LAN. Figure 37 on page 59 illustrates the response time when using a guest LAN versus a VSWITCH LAN.

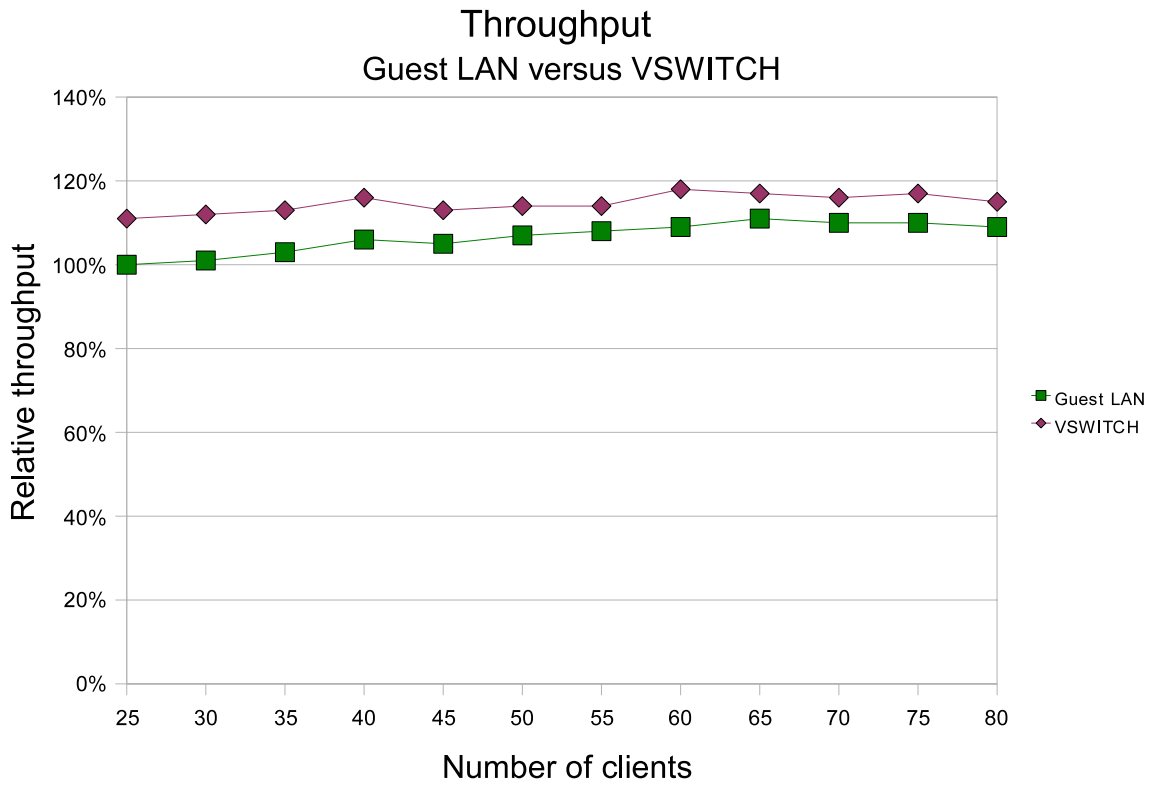

Figure 36. Throughput Comparison - guest LAN versus VSWITCH LAN.

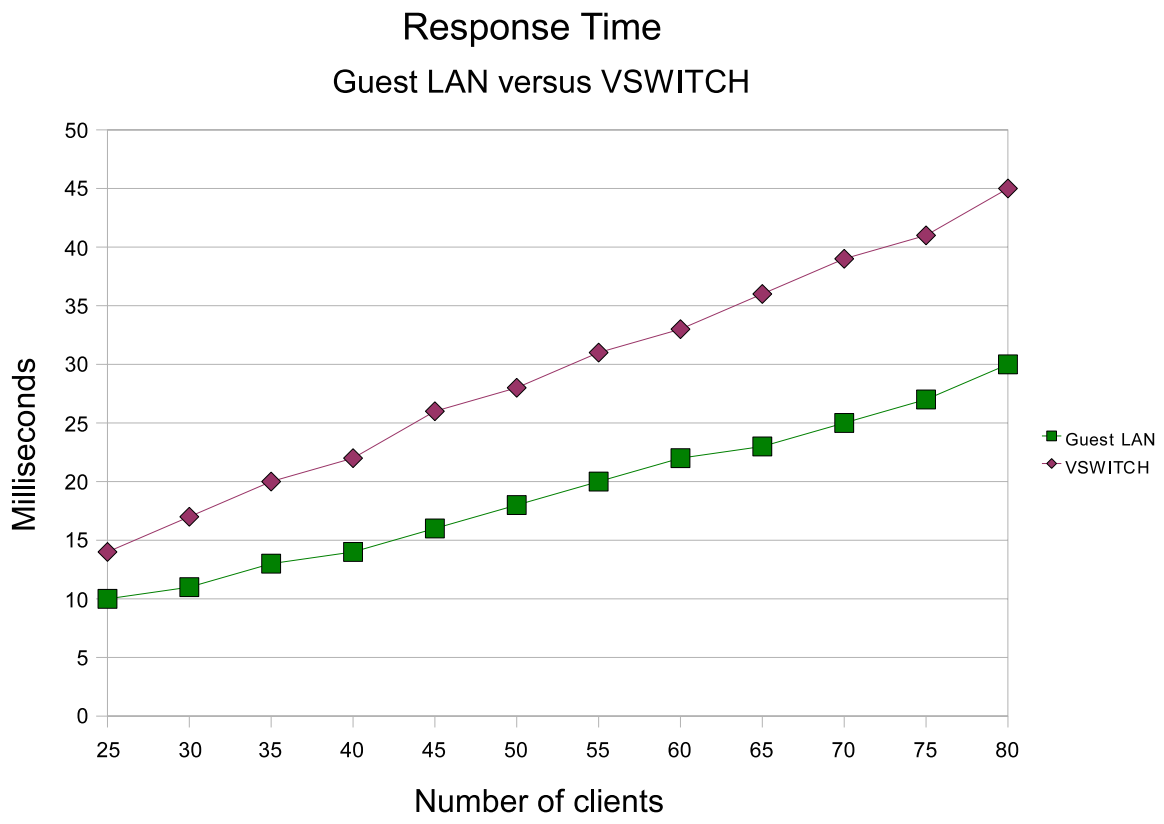

Figure 37. Response Time Comparison - guest LAN versus VSWITCH LAN.

Figure 38 on page 60 illustrates the CPU utilization when using a guest LAN versus a VSWITCH LAN.

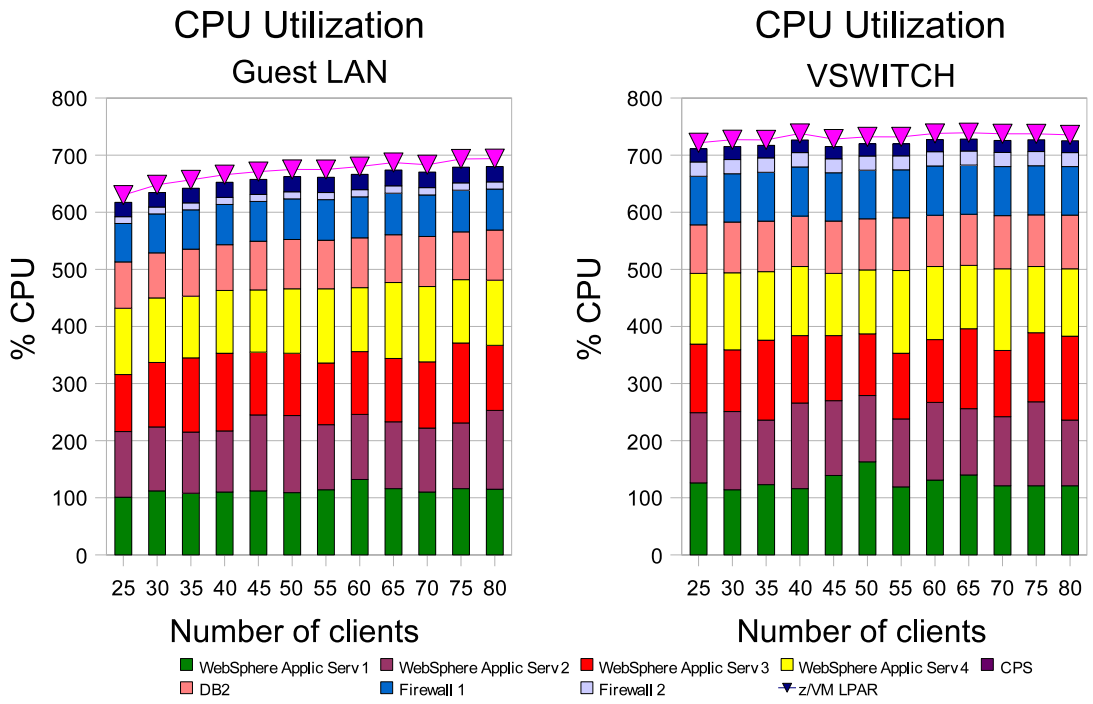

Figure 38. CPU comparison - guest LAN versus VSWITCH LAN

Observations

Configuring a z/VM VSWITCH LAN produced an improvement of approximately 8% in throughput and an increase in response time. Total CPU utilization for the VSWITCH case is higher than the guest LAN case.

Figure 39 on page 61 shows where the additional CPU utilization is spent. There is significant increase in WebSphere CPU utilization. The total CPU used increased up to 10%, which indicates in relation to the 8% throughput improvement, a slight increase in CPU cost per transaction.

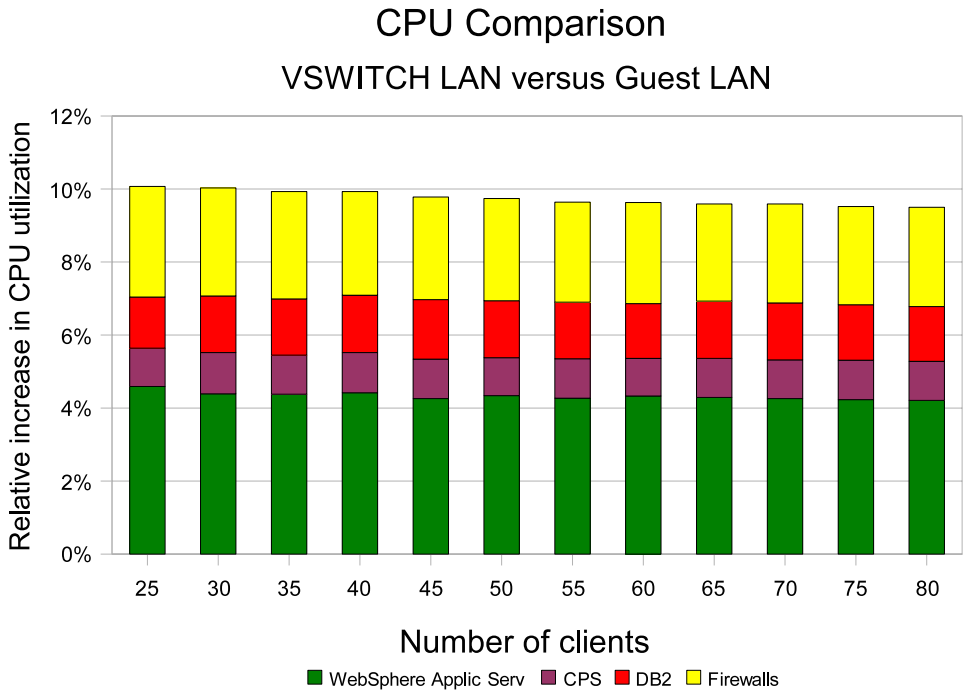

Figure 39. CPU utilization comparison - VSWITCH versus guest LAN

Note: The values were stacked to show the increase of the utilization in total.

Conclusion

Whenever possible, the z/VM VSWITCH function should be used instead of the z/VM guest LAN function.

Detailed setup examples

These are detailed examples of configurations and sample scripts used in the test runs.

web.xml setup examples

This is the `web.xml` configuration file used to set the configuration parameters for Trade. The following stanza is set for the various caching modes used.

For no caching:

```
<init-param id="InitParam_10">
    <description>Sets the data caching type</description>
    <param-name>CachingType</param-name>
    <param-value>No Caching</param-value>
 </init-param>
```

For Distributed Map:

```
<param-value>DistributedMap</param-value>
```

For command caching:

```
<param-value>Command Caching</param-value>
```

The complete configuration file:

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app id="WebApp_1" version="2.4"
 xmlns="http://java.sun.com/xml/ns/j2ee"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee
http://java.sun.com/xml/ns/j2ee/web-app_2_4.xsd">

  <display-name>TradeWeb</display-name>
  <filter id="Filter_1">
   <display-name>OrdersAlertFilter</display-name>
   <filter-name>OrdersAlertFilter</filter-name>
 <filter-class>com.ibm.websphere.samples.trade.web.OrdersAlertFilter
      </filter-class>
  </filter>
  <filter-mapping id="FilterMapping_1">
   <filter-name>OrdersAlertFilter</filter-name>
   <servlet-name>TradeAppServlet</servlet-name>
  </filter-mapping>
  <listener id="Listener_1">
<listener-class>com.ibm.websphere.samples.trade.web.TradeWebContextListener
     </listener-class>
  </listener>
  <servlet id="Servlet_1">
   <display-name>TradeAppServlet</display-name>
   <servlet-name>TradeAppServlet</servlet-name>
   <servlet-class>com.ibm.websphere.samples.trade.web.TradeAppServlet
        </servlet-class>
 <load-on-startup>1</load-on-startup>
  </servlet>
  <servlet id="Servlet_2">
   <display-name>register</display-name>
   <servlet-name>register</servlet-name>
   <jsp-file>/register.jsp</jsp-file>
  </servlet>
     <servlet id="Servlet_3">
   <display-name>TestServlet</display-name>
   <servlet-name>TestServlet</servlet-name>
   <servlet-class>com.ibm.websphere.samples.trade.web.TestServlet
      </servlet-class>
  </servlet>
  <servlet id="Servlet_4">
   <display-name>welcome</display-name>
   <servlet-name>welcome</servlet-name>
   <jsp-file>/welcome.jsp</jsp-file>
  </servlet>
  <servlet id="Servlet_5">
   <display-name>order</display-name>
   <servlet-name>order</servlet-name>
   <jsp-file>/order.jsp</jsp-file>
  </servlet>
  <servlet id="Servlet_6">
   <display-name>tradehome</display-name>
   <servlet-name>tradehome</servlet-name>
   <jsp-file>/tradehome.jsp</jsp-file>
  </servlet>
  <servlet id="Servlet_7">
   <display-name>TradeConfigServlet</display-name>
   <servlet-name>TradeConfigServlet</servlet-name>
<servlet-class>com.ibm.websphere.samples.trade.web.TradeConfigServlet
      </servlet-class>
  </servlet>
  <servlet id="Servlet_8">
   <display-name>TradeScenarioServlet</display-name>
   <servlet-name>TradeScenarioServlet</servlet-name>
<servlet-class>com.ibm.websphere.samples.trade.web.TradeScenarioServlet
        </servlet-class>
   <init-param id="InitParam_1">
```

```
    <description>Sets the default RuntimeMode. Legal values
     include EJB and Direct</description>
    <param-name>runTimeMode</param-name>
    <param-value>EJB</param-value>
   </init-param>
   <init-param id="InitParam_2">
    <description>Sets the default Order Processing Mode. Legal
 values include Synchronous, Asynchronous_1-Phase and Asynchronous_2-Phase
          </description>
    <param-name>orderProcessingMode</param-name>
    <param-value>Synchronous</param-value>
   </init-param>
   <init-param id="InitParam_3">
    <description>Sets the protocol the web application
communicates with the server side services when driving
     with TradeScenarioServlet. Legal values include Standard and WebServices.
           </description>
    <param-name>accessMode</param-name>
    <param-value>Standard</param-value>
   </init-param>
<!-- Commented out as this will only work if IHS is installed, unless the \
   default is changed to port 9080
  <init-param id="InitParam_4">
   <description>Sets the WebServices endpoint when using
    WebServices accessMode when driving with TradeScenarioServlet.
          </description>
   <param-name>webServicesEndpoint</param-name>
   <param-value>http://localhost/trade/services/TradeWSServices?wsdl</param-value>
  </init-param>
  <init-param id="InitParam_5">
   <description>Sets the default workloadMix used with
    TradeScenario servlet. Legal values include Standard and High-Volume
         </description>

     <param-name>workloadMix</param-name>
   <param-value>Standard</param-value>
  </init-param>
  <init-param id="InitParam_6">
   <description>Sets the default WebInterface. Legal values
    include JSP and JSP-images</description>
   <param-name>WebInterface</param-name>
   <param-value>JSP</param-value>
  </init-param>
  <init-param id="InitParam_7">
   <description>Sets the population of Trade users when driving
    with TradeScenarioServlet.</description>
   <param-name>maxUsers</param-name>
   <param-value>2000</param-value>
  </init-param>
  <init-param id="InitParam_8">
   <description>Sets the population of Stock quotes used when
    driving with TradeScenarioServlet.</description>
   <param-name>maxQuotes</param-name>
   <param-value>1000</param-value>
  </init-param>
  <init-param id="InitParam_9">
   <description>Sets the number of iterations on web/ejb
    primitives.</description>
   <param-name>primIterations</param-name>
   <param-value>1</param-value>
  </init-param>
  <init-param id="InitParam_10">
   <description>Sets the data caching type</description>
   <param-name>CachingType</param-name>
   <param-value>No Caching</param-value>
  </init-param>
  <init-param id="InitParam_11">
```

```
  <description>Enables or disables the long run mode</description>
  <param-name>LongRun</param-name>
  <param-value>true</param-value>
 </init-param>
  <load-on-startup>1</load-on-startup>
 </servlet>
 <servlet id="Servlet_9">
  <display-name>PingServlet</display-name>
  <servlet-name>PingServlet</servlet-name>
  <servlet-class>com.ibm.websphere.samples.trade.web.prims.PingServlet
      </servlet-class>
 </servlet>
 <servlet id="Servlet_10">
  <display-name>PingServletWriter</display-name>
  <servlet-name>PingServletWriter</servlet-name>
  <servlet-class>com.ibm.websphere.samples.trade.web.prims.PingServletWriter
      </servlet-class>
 </servlet>
 <servlet id="Servlet_11">
  <display-name>PingServlet2Servlet</display-name>
  <servlet-name>PingServlet2Servlet</servlet-name>
  <servlet-class>com.ibm.websphere.samples.trade.web.prims.PingServlet2Servlet
     </servlet-class>
 </servlet>
 <servlet id="Servlet_12">
  <display-name>PingServlet2ServletRcv</display-name>
  <servlet-name>PingServlet2ServletRcv</servlet-name>
<servlet-class>com.ibm.websphere.samples.trade.web.prims.PingServlet2ServletRcv
   </servlet-class>
 </servlet>
 <servlet id="Servlet_13">
  <display-name>PingServlet2Jsp</display-name>
  <servlet-name>PingServlet2Jsp</servlet-name>
  <servlet-class>com.ibm.websphere.samples.trade.web.prims.PingServlet2Jsp
     </servlet-class>
 </servlet>
 <servlet id="Servlet_14">
  <display-name>PingSession1</display-name>
  <servlet-name>PingSession1</servlet-name>
  <servlet-class>com.ibm.websphere.samples.trade.web.prims.PingSession1
     </servlet-class>
 </servlet>
 <servlet id="Servlet_15">
  <display-name>PingSession2</display-name>
  <servlet-name>PingSession2</servlet-name>
  <servlet-class>com.ibm.websphere.samples.trade.web.prims.PingSession2
     </servlet-class>
 </servlet>
 <servlet id="Servlet_16">
  <display-name>PingSession3</display-name>
  <servlet-name>PingSession3</servlet-name>
  <servlet-class>com.ibm.websphere.samples.trade.web.prims.PingSession3
     </servlet-class>
 </servlet>
 <servlet id="Servlet_17">
  <display-name>PingJDBCRead</display-name>
  <servlet-name>PingJDBCRead</servlet-name>
  <servlet-class>com.ibm.websphere.samples.trade.web.prims.PingJDBCRead
     </servlet-class>
 </servlet>
 <servlet id="Servlet_18">
  <display-name>PingJDBCWrite</display-name>
  <servlet-name>PingJDBCWrite</servlet-name>
  <servlet-class>com.ibm.websphere.samples.trade.web.prims.PingJDBCWrite
     </servlet-class>
 </servlet>
 <servlet id="Servlet_19">
```

```
   <display-name>PingServlet2Session</display-name>
   <servlet-name>PingServlet2Session</servlet-name>
   <servlet-class>com.ibm.websphere.samples.trade.web.prims.PingServlet2Session
      </servlet-class>
  </servlet>
<servlet id="Servlet_20">
   <display-name>PingServlet2EntityLocal</display-name>
   <servlet-name>PingServlet2EntityLocal</servlet-name>
 <servlet-class>com.ibm.websphere.samples.trade.web.prims.PingServlet2EntityLocal
    </servlet-class>
  </servlet>
  <servlet id="Servlet_29">
   <display-name>PingServlet2EntityRemote</display-name>
   <servlet-name>PingServlet2EntityRemote</servlet-name>
 <servlet-class>com.ibm.websphere.samples.trade.web.prims.PingServlet2EntityRemote
    </servlet-class>
  </servlet>
  <servlet id="Servlet_21">
   <display-name>PingServlet2Session2Entity</display-name>
   <servlet-name>PingServlet2Session2Entity</servlet-name>
  <servlet-class>com.ibm.websphere.samples.trade.web.prims.PingServlet2Session2Entity
    </servlet-class>
  </servlet>
  <servlet id="Servlet_22">
   <display-name>PingServlet2Session2EntityCollection</display-name>
   <servlet-name>PingServlet2Session2EntityCollection</servlet-name>
 <servlet-class>com.ibm.websphere.samples.trade.web.prims.PingServlet2Session2EntityCollection
    </servlet-class>
  </servlet>
  <servlet id="Servlet_23">
   <display-name>PingServlet2Session2CMROne2One</display-name>
     <servlet-name>PingServlet2Session2CMROne2One</servlet-name>
 <servletclass>com.ibm.websphere.samples.trade.web.prims.PingServlet2Session2CMROne2One
    </servlet-class>
  </servlet>
  <servlet id="Servlet_24">
   <display-name>PingServlet2Session2CMROne2Many</display-name>
   <servlet-name>PingServlet2Session2CMROne2Many</servlet-name>
 <servlet-class>com.ibm.websphere.samples.trade.web.prims.PingServlet2Session2CMROne2Many
    </servlet-class>
  </servlet>
  <servlet id="Servlet_25">
   <display-name>PingServlet2MDBQueue</display-name>
    <servlet-name>PingServlet2MDBQueue</servlet-name>
<servlet-class>com.ibm.websphere.samples.trade.web.prims.PingServlet2MDBQueue</servlet-class>
  </servlet>
  <servlet id="Servlet_26">
   <display-name>PingServlet2MDBTopic</display-name>
   <servlet-name>PingServlet2MDBTopic</servlet-name>
   <servlet-class>com.ibm.websphere.samples.trade.web.prims.PingServlet2MDBTopic
      </servlet-class>
  </servlet>
  <servlet id="Servlet_27">
   <display-name>PingServlet2JNDI</display-name>
   <servlet-name>PingServlet2JNDI</servlet-name>
   <servlet-class>com.ibm.websphere.samples.trade.web.prims.PingServlet2JNDI
      </servlet-class>
  </servlet>
  <servlet id="Servlet_28">
   <display-name>PingServlet2TwoPhase</display-name>
   <servlet-name>PingServlet2TwoPhase</servlet-name>
   <servlet-class>com.ibm.websphere.samples.trade.web.prims.PingServlet2TwoPhase
      </servlet-class>
  </servlet>
```

```
<servlet id="Servlet_31">
  <display-name>PingServlet2Include</display-name>
  <servlet-name>PingServlet2Include</servlet-name>
  <servlet-class>com.ibm.websphere.samples.trade.web.prims.PingServlet2Include
     </servlet-class>
 </servlet>
 <servlet id="Servlet_32">
  <display-name>PingServlet2IncludeRcv</display-name>
  <servlet-name>PingServlet2IncludeRcv</servlet-name>
  <servlet-class>com.ibm.websphere.samples.trade.web.prims.PingServlet2IncludeRcv
   </servlet-class>
 </servlet>
 <servlet id="Servlet_30">
  <display-name>com_ibm_websphere_samples_trade_TradeWSAction
       </display-name>
  <servlet-name>com_ibm_websphere_samples_trade_TradeWSAction
       </servlet-name>
  <servlet-class>com.ibm.websphere.samples.trade.TradeWSAction
     </servlet-class>
 </servlet>
 <servlet-mapping id="ServletMapping_1">
  <servlet-name>TradeAppServlet</servlet-name>
  <url-pattern>/app</url-pattern>
 </servlet-mapping>
 <servlet-mapping id="ServletMapping_2">
  <servlet-name>TestServlet</servlet-name>
  <url-pattern>/TestServlet</url-pattern>
 </servlet-mapping>
 <servlet-mapping id="ServletMapping_3">
  <servlet-name>TradeConfigServlet</servlet-name>
  <url-pattern>/config</url-pattern>
 </servlet-mapping>
 <servlet-mapping id="ServletMapping_4">
  <servlet-name>TradeScenarioServlet</servlet-name>
  <url-pattern>/scenario</url-pattern>
 </servlet-mapping>
 <servlet-mapping id="ServletMapping_5">
  <servlet-name>PingServlet</servlet-name>
  <url-pattern>/servlet/PingServlet</url-pattern>
 </servlet-mapping>
 <servlet-mapping id="ServletMapping_6">
  <servlet-name>PingServletWriter</servlet-name>
  <url-pattern>/servlet/PingServletWriter</url-pattern>
 </servlet-mapping>
 <servlet-mapping id="ServletMapping_7">
  <servlet-name>PingServlet2Servlet</servlet-name>
  <url-pattern>/servlet/PingServlet2Servlet</url-pattern>
 </servlet-mapping>
 <servlet-mapping id="ServletMapping_8">
  <servlet-name>PingServlet2ServletRcv</servlet-name>
  <url-pattern>/servlet/PingServlet2ServletRcv</url-pattern>
 </servlet-mapping>
 <servlet-mapping id="ServletMapping_9">
  <servlet-name>PingServlet2Jsp</servlet-name>
  <url-pattern>/servlet/PingServlet2Jsp</url-pattern>
 </servlet-mapping>
 <servlet-mapping id="ServletMapping_10">
  <servlet-name>PingSession1</servlet-name>
  <url-pattern>/servlet/PingSession1</url-pattern>
 </servlet-mapping>
 <servlet-mapping id="ServletMapping_11">
  <servlet-name>PingSession2</servlet-name>
  <url-pattern>/servlet/PingSession2</url-pattern>
 </servlet-mapping>
 <servlet-mapping id="ServletMapping_12">
  <servlet-name>PingSession3</servlet-name>
  <url-pattern>/servlet/PingSession3</url-pattern>
```

```
</servlet-mapping>
<servlet-mapping id="ServletMapping_13">
 <servlet-name>PingJDBCRead</servlet-name>
 <url-pattern>/servlet/PingJDBCRead</url-pattern>
</servlet-mapping>
<servlet-mapping id="ServletMapping_14">
 <servlet-name>PingJDBCWrite</servlet-name>
 <url-pattern>/servlet/PingJDBCWrite</url-pattern>
</servlet-mapping>
<servlet-mapping id="ServletMapping_15">
 <servlet-name>PingServlet2Session</servlet-name>
 <url-pattern>/servlet/PingServlet2Session</url-pattern>
</servlet-mapping>
<servlet-mapping id="ServletMapping_16">
 <servlet-name>PingServlet2EntityLocal</servlet-name>
 <url-pattern>/servlet/PingServlet2EntityLocal</url-pattern>
</servlet-mapping>
<servlet-mapping id="ServletMapping_25">
 <servlet-name>PingServlet2EntityRemote</servlet-name>
 <url-pattern>/servlet/PingServlet2EntityRemote</url-pattern>
</servlet-mapping>
<servlet-mapping id="ServletMapping_17">
 <servlet-name>PingServlet2Session2Entity</servlet-name>
 <url-pattern>/servlet/PingServlet2Session2Entity</url-pattern>
</servlet-mapping>
<servlet-mapping id="ServletMapping_18">
 <servlet-name>PingServlet2Session2EntityCollection</servlet-name>
 <url-pattern>/servlet/PingServlet2Session2EntityCollection</url-pattern>
</servlet-mapping>
<servlet-mapping id="ServletMapping_19">
 <servlet-name>PingServlet2Session2CMROne2One</servlet-name>
 <url-pattern>/servlet/PingServlet2Session2CMROne2One</url-pattern>
</servlet-mapping>
<servlet-mapping id="ServletMapping_20">
 <servlet-name>PingServlet2Session2CMROne2Many</servlet-name>
 <url-pattern>/servlet/PingServlet2Session2CMROne2Many</url-pattern>
</servlet-mapping>
<servlet-mapping id="ServletMapping_21">
 <servlet-name>PingServlet2MDBQueue</servlet-name>
 <url-pattern>/servlet/PingServlet2MDBQueue</url-pattern>
</servlet-mapping>
<servlet-mapping id="ServletMapping_22">
 <servlet-name>PingServlet2MDBTopic</servlet-name>
 <url-pattern>/servlet/PingServlet2MDBTopic</url-pattern>
</servlet-mapping>
<servlet-mapping id="ServletMapping_23">
 <servlet-name>PingServlet2JNDI</servlet-name>
 <url-pattern>/servlet/PingServlet2JNDI</url-pattern>
</servlet-mapping>
<servlet-mapping id="ServletMapping_24">
 <servlet-name>PingServlet2TwoPhase</servlet-name>
 <url-pattern>/servlet/PingServlet2TwoPhase</url-pattern>
</servlet-mapping>
 <servlet-mapping id="ServletMapping_26">
 <servlet-name>PingServlet2Include</servlet-name>
 <url-pattern>/servlet/PingServlet2Include</url-pattern>
</servlet-mapping>
<session-config id="SessionConfig_1">
 <session-timeout>30</session-timeout>
</session-config>
<welcome-file-list id="WelcomeFileList_1">
 <welcome-file>index.html</welcome-file>
</welcome-file-list>
<error-page id="ExceptionTypeErrorPage_1">
 <exception-type>java.lang.Exception</exception-type>
 <location>/error.jsp</location>
</error-page>
```

```
<error-page id="ErrorCodeErrorPage_1">
 <error-code>500</error-code>
 <location>/error.jsp</location>
</error-page
<message-destination-ref id="MessageDestinationRef_1">
 <message-destination-ref-name>jms/TradeBrokerQueue
        </message-destination-ref-name>
 <message-destination-type>javax.jms.Queue
        </message-destination-type>
 <message-destination-usage>Produces
        </message-destination-usage>
 <message-destination-link>tradeEJB.jar#TradeBrokerQueue
        </message-destination-link>
</message-destination-ref>
<message-destination-ref id="MessageDestinationRef_2">
 <message-destination-ref-name>jms/TradeStreamerTopic
        </message-destination-ref-name>
 <message-destination-type>javax.jms.Topic
        </message-destination-type>
 <message-destination-usage>Produces
        </message-destination-usage>
 <message-destination-link>tradeEJB.jar#TradeStreamerTopic
      </message-destination-link>
</message-destination-ref>
<resource-ref id="ResourceRef_1">
 <res-ref-name>jdbc/TradeDataSource</res-ref-name>
 <res-type>javax.sql.DataSource</res-type>
 <res-auth>Container</res-auth>
 <res-sharing-scope>Shareable</res-sharing-scope>
</resource-ref>
<resource-ref id="ResourceRef_2">
 <res-ref-name>jms/QueueConnectionFactory</res-ref-name>
 <res-type>javax.jms.QueueConnectionFactory</res-type>
 <res-auth>Application</res-auth>
 <res-sharing-scope>Shareable</res-sharing-scope>
</resource-ref>
<resource-ref id="ResourceRef_3">
 <res-ref-name>jms/TopicConnectionFactory</res-ref-name>
 <res-type>javax.jms.TopicConnectionFactory</res-type>
 <res-auth>Application</res-auth>
 <res-sharing-scope>Shareable</res-sharing-scope>
</resource-ref>

<ejb-ref id="EjbRef_1">
 <ejb-ref-name>ejb/Trade</ejb-ref-name>
 <ejb-ref-type>Session</ejb-ref-type>
 <home>com.ibm.websphere.samples.trade.ejb.TradeHome</home>
 <remote>com.ibm.websphere.samples.trade.ejb.Trade</remote>
 <ejb-link>TradeEJB</ejb-link>
</ejb-ref>
<ejb-ref id="EjbRef_2">
 <ejb-ref-name>ejb/Quote</ejb-ref-name>
 <ejb-ref-type>Entity</ejb-ref-type>
 <home>com.ibm.websphere.samples.trade.ejb.QuoteHome</home>
 <remote>com.ibm.websphere.samples.trade.ejb.Quote</remote>
 <ejb-link>QuoteEJB</ejb-link>
</ejb-ref>
<ejb-local-ref id="EJBLocalRef_1">
 <ejb-ref-name>ejb/LocalQuote</ejb-ref-name>
 <ejb-ref-type>Entity</ejb-ref-type>
 <local-home>com.ibm.websphere.samples.trade.ejb.LocalQuoteHome
       </local-home>
 <local>com.ibm.websphere.samples.trade.ejb.LocalQuote</local>
 <ejb-link>QuoteEJB</ejb-link>
</ejb-local-ref>
<ejb-local-ref id="EJBLocalRef_2">
 <ejb-ref-name>ejb/LocalAccountHome</ejb-ref-name>
 <ejb-ref-type>Entity</ejb-ref-type>
```

```
   <local-home>com.ibm.websphere.samples.trade.ejb.LocalAccountHome
         </local-home>
   <local>com.ibm.websphere.samples.trade.ejb.LocalAccount</local>
   <ejb-link>AccountEJB</ejb-link>
  </ejb-local-ref>
  <service-ref>
   <description>WSDL Service Trade</description>
   <service-ref-name>service/Trade</service-ref-name>
   <service-interface>com.ibm.websphere.samples.trade.client.ws.Trade
        </service-interface>
   <wsdl-file>WEB-INF/wsdl/TradeServices.wsdl</wsdl-file>
   <jaxrpc-mapping-file>WEB-INF/TradeServicesClient_mapping.xml
         </jaxrpc-mapping-file>
   <service-qname xmlns:pfx="http://trade.samples.websphere.ibm.com">pfx:Trade
         </service-qname>
   <port-component-ref>
    <service-endpoint-interface>
     com.ibm.websphere.samples.trade.client.ws.TradeWSServices
         </service-endpoint-interface>
   </port-component-ref>
  </service-ref>
</web-app>
```

cachespec.xml setup examples

This is the cachespec.xml configuration file used to set the configuration parameters for Trade. The following stanza is set for the various caching modes used.

```
<?xml version="1.0"?>

<!DOCTYPE cache SYSTEM "cachespec.dtd">


<!--

   DynaCache specification for WebSphere performance application.
   Trade provides a sample application for DynaCache usage and configuration.
  -->
<cache>
    <cache-entry>
    <class>servlet</class>
    <name>/app</name>


<!-- Cacheid and dependency-id for the Portfolio page. This page is invalidated
when a stock buy/sell order has been

     completed. -->
     <cache-id>
       <component id="action" type="parameter">
         <value>portfolio</value>
         <required>true</required>
       </component>
       <component id="JSESSIONID" type="cookie">
         <required>true</required>
       </component>
         <property name="consume-subfragments">true</property>
       <property name="EdgeCacheable">true</property>
     </cache-id>
     <!-- Special case for TradeScenarioServlet to invoke Portfolio and DynaCache
  not to return an ESI include
      but actually invoke the portfolio operation
     -->
     <cache-id>
       <component id="action" type="parameter">
         <value>portfolioNoEdge</value>
```

```
        <required>true</required>
      </component>
      <component id="JSESSIONID" type="cookie">
        <required>true</required>
      </component>
       <property name="consume-subfragments">true</property>
      <property name="EdgeCacheable">false</property>
    </cache-id>

  <!-- Dependency-id for the portfolio page -->
    <dependency-id>Holdings_UserID
     <component id="action" type="parameter" ignore-value="true">
        <value>portfolio</value>
        <required>true</required>
      </component>
      <component id="uidBean" type="session">
        <required>true</required>
      </component>
    </dependency-id>
  <!-- This dependency id is to invalidate the portfolio page when the user
 is logged out.  The account and
    home pages are invalidated when the user is logged out based on the
dependency id Account_UserID -->

    <dependency-id>Holdings_UserID1
     <component id="action" type="parameter" ignore-value="true">
        <value>portfolio</value>
        <required>true</required>
      </component>
      <component id="uidBean" type="session">
        <required>true</required>
      </component>
    </dependency-id>
  <dependency-id>Holdings_UserID
      <component id="action" type="parameter" ignore-value="true">
        <value>portfolioNoEdge</value>
        <required>true</required>
      </component>
      <component id="uidBean" type="session">
        <required>true</required>
      </component>
    </dependency-id>
<!-- Cacheid and dependency id for the Account page. This page is invalidated
when the account profile has been updated or
    a stock buy/sell order has been completed. -->
    <cache-id>
      <component id="action" type="parameter">
        <value>account</value>
        <required>true</required>
      </component>
      <component id="JSESSIONID" type="cookie">
        <required>true</required>
      </component>
       <property name="consume-subfragments">true</property>
      <property name="EdgeCacheable">true</property>
    </cache-id>

     <!-- Dependency-id for the account page -->
    <dependency-id>Account_UserID
     <component id="action" type="parameter" ignore-value="true">
        <value>account</value>
        <required>true</required>
      </component>
      <component id="uidBean" type="session">
        <required>true</required>
      </component>
    </dependency-id>
```

```
     <dependency-id>AccountProfile_UserID
      <component id="action" type="parameter" ignore-value="true">
         <value>account</value>
         <required>true</required>
       </component>
       <component id="uidBean" type="session">
         <required>true</required>
       </component>
     </dependency-id>

<!-- Cache id for the Trade Home page. This page is invalidated when ever there
 is a change in the account information or
   the holdings information -->
     <cache-id>
       <component id="action" type="parameter">
         <value>home</value>
         <required>true</required>
       </component>
       <component id="JSESSIONID" type="cookie">
         <required>true</required>
       </component>
       <property name="consume-subfragments">true</property>
       <property name="EdgeCacheable">true</property>
     </cache-id>
     <dependency-id>Account_UserID
      <component id="action" type="parameter" ignore-value="true">
         <value>home</value>
         <required>true</required>
       </component>
       <component id="uidBean" type="session">
         <required>true</required>
       </component>
     </dependency-id>
     <dependency-id>Holdings_UserID
      <component id="action" type="parameter" ignore-value="true">
         <value>home</value>
         <required>true</required>
       </component>
       <component id="uidBean" type="session">
         <required>true</required>
       </component>
     </dependency-id>

<!-- Cache entry for the quotes page. This page will be invalidated whenever
 a quote of a symbol is invalidated -->
     <cache-id>
       <component id="action" type="parameter">
         <value>quotes</value>
         <required>true</required>
       </component>
       <component id="symbols" type="parameter">
         <required>true</required>
       </component>
       <property name="consume-subfragments">true</property>
       <property name="EdgeCacheable">true</property>
     </cache-id>
    <dependency-id>AllQuotes
       <component id="action" type="parameter" ignore-value="true">
         <value>quotes</value>
         <required>true</required>
       </component>
     </dependency-id>
    <dependency-id>Quote_Symbol
       <component id="action" type="parameter" ignore-value="true">
         <value>quotes</value>
         <required>true</required>
```

```
      </component>
    </dependency-id>
  </cache-entry>


<!-- Cache entry for register.jsp which is included in the Welcome.jsp -->
  <cache-entry>
     <class>servlet</class>
     <name>/register.jsp</name>
      <property name="consume-subfragments">true</property>
     <property name="EdgeCacheable">true</property>
     <cache-id>
       <timeout>180</timeout>
     </cache-id>
  </cache-entry>

<!-- Cache entry for marketSummary.jsp. This jsp gets invalidated on a timeout
 as well as when Trade database is reset -->
  <cache-entry>
     <class>servlet</class>
     <name>/marketSummary.jsp</name>
      <property name="consume-subfragments">true</property>
     <property name="EdgeCacheable">true</property>
     <cache-id>
       <priority>3</priority>
       <timeout>180</timeout>
     </cache-id>
     <dependency-id>MarketSummary
     </dependency-id>
  </cache-entry>

<!-- Cache entry for displayQuote.jsp. This page is invalidated for each
individual symbol when that symbol is updated with the updateQuotePriceCommand -->

  <cache-entry>
     <class>servlet</class>
     <name>/displayQuote.jsp</name>
      <property name="consume-subfragments">true</property>
     <property name="EdgeCacheable">true</property>
     <cache-id>
       <component id="symbol" type="parameter">
         <required>true</required>
       </component>
       <priority>3</priority>
     </cache-id>

  <!-- This dependency id is setup to identify stocks by their symbol
     -->
     <dependency-id>Quote_Symbol
       <component id="symbol" type="parameter">
         <required>true</required>
       </component>
     </dependency-id>

  <!-- This dependency id is setup to identify all Quotes
      for commands which invalidate all Quote entries
     -->
     <dependency-id>AllQuotes
     </dependency-id>
   </cache-entry>

<!--

 The MarketSummaryCommand is cached and invalidated on a timeout.
 Time based invalidation is a simple, yet common usage for caching.
   -->
   <cache-entry>
```

```
      <class>command</class>
      <sharing-policy>not-shared</sharing-policy>
      <name>com.ibm.websphere.samples.trade.command.MarketSummaryCommand
              </name>
      <cache-id>
        <priority>3</priority>
        <timeout>10</timeout>
      </cache-id>
      <dependency-id>MarketSummary
      </dependency-id>
    </cache-entry>

<!--
 The QuoteCommand is used to cache Quote symbols and prices.
 QuoteCommands are invalidated for each individual symbol when that
 symbol is updated with the UpdateQuotePriceCommand
   -->

    <cache-entry>
      <class>command</class>
      <sharing-policy>not-shared</sharing-policy>
      <name>com.ibm.websphere.samples.trade.command.QuoteCommand
             </name>
      <cache-id>
        <component type="method" id="getSymbol">
          <required>true</required>
        </component>
        <priority>3</priority>
      </cache-id>

   <!-- This dependency id is setup to identify stocks by their symbol
      -->
      <dependency-id>Quote_Symbol
        <component id="getSymbol" type="method">
          <required>true</required>
        </component>
      </dependency-id>

   <!-- This dependency id is setup to identify all Quotes
        for commands which invalidate all Quote entries
      -->
      <dependency-id>AllQuotes
      </dependency-id>
    </cache-entry>

<!--
    The UpdateQuotePriceCommand is used to update the current price for
    a stock symbol. This invalidates the QuoteCommand cache-entry above
    for the individual stock, identified by getSymbol. The getSymbol method
    ties this entry to the dependency id for the QuoteCommand cache-entry above
    -->
    <cache-entry>
      <class>command</class>
      <sharing-policy>not-shared</sharing-policy>
      <name>com.ibm.websphere.samples.trade.command.UpdateQuotePriceCommand</name>

      <invalidation>Quote_Symbol
        <component id="getSymbol" type="method">
          <required>true</required>
        </component>
      </invalidation>
    </cache-entry>

<!--
    The AccountCommand is used to cache a user's Account information.
    AccountCommands are invalidated for each individual user when that
    user's account information, such as their account balance,
```

```
   is updated by a trading operation
   -->

   <cache-entry>
     <class>command</class>
     <sharing-policy>not-shared</sharing-policy>
     <name>com.ibm.websphere.samples.trade.command.AccountCommand</name>
     <cache-id>
       <component type="method" id="getUserID">
         <required>true</required>
       </component>
       <priority>3</priority>
     </cache-id>

     <!-- This dependency id is setup to identify user Accounts by the userID
     -->
     <dependency-id>Account_UserID
       <component id="getUserID" type="method">
         <required>true</required>
       </component>
     </dependency-id>

     <!-- This dependency id is setup to identify all user Accounts
     for commands which invalidate all Account cache entries
     -->
     <dependency-id>AllUsers
     </dependency-id>
   </cache-entry>

<!--
   The AccountProfileCommand is used to cache a user's Account profile information.
   AccountProfileCommands are invalidated for each individual user when that
   user's account profile information is updated by the UpdateAccountProfileCommand
   -->

   <cache-entry>
     <class>command</class>
     <sharing-policy>not-shared</sharing-policy>
     <name>com.ibm.websphere.samples.trade.command.AccountProfileCommand</name>
     <cache-id>
       <component type="method" id="getUserID">
         <required>true</required>
       </component>
       <priority>3</priority>
     </cache-id>

     <!-- This dependency id is setup to identify user Account Profiles by
     the userID -->
     <dependency-id>AccountProfile_UserID
       <component id="getUserID" type="method">
         <required>true</required>
       </component>
     </dependency-id>

     <!-- This dependency id is setup to identify all user Account
       for commands which invalidate all Account Profile cache entries
     -->
     <dependency-id>AllUsers
     </dependency-id>

   </cache-entry>

<!--

   The HoldingsCommand is used to cache a user's Stock holdings information.
   HoldingCommands are invalidated for each individual user when that
   user's holdings change due to an OrderCompletedCommand
```

```
   -->
   <cache-entry>
     class>command</class>
     <sharing-policy>not-shared</sharing-policy>
     <name>com.ibm.websphere.samples.trade.command.HoldingsCommand</name>
     <cache-id>
       <component type="method" id="getUserID">
         <required>true</required>
       </component>
       <priority>3</priority>
     </cache-id>

     <!-- This dependency id is setup to identify user Accounts by the userID
     -->
     <dependency-id>Holdings_UserID
       <component id="getUserID" type="method">
         <required>true</required>
       </component>
     </dependency-id>

     <!-- This dependency id is setup to identify all user Accounts
     for commands which invalidate all Account Holdings cache entries
     -->
     <dependency-id>AllUsers
     </dependency-id>

   </cache-entry>

<!--
   The OrdersCommand is used to cache a user's Orders information.
   OrdersCommands are invalidated for each individual user when that
   user's enters a new order with a BuyCommand or SellCommand.
   The OrdersCommand is also invalidated when an order status is changed
   by the OrderCompletedCommand
   -->
   <cache-entry>
     <class>command</class>
     <sharing-policy>not-shared</sharing-policy>
     <name>com.ibm.websphere.samples.trade.command.OrdersCommand</name>
     <cache-id>
       <component type="method" id="getUserID">
         <required>true</required>
       </component>
       <priority>3</priority>
     </cache-id>

     <!-- The dependency id is setup to identify user Accounts by the userID
     -->
     <dependency-id>Orders_UserID
       <component id="getUserID" type="method">
         <required>true</required>
       </component>
     </dependency-id>

     <!-- This dependency id is setup to identify all user Accounts
     for commands which invalidate all Account Orders cache entries
     -->
     <dependency-id>AllUsers
     </dependency-id>

   </cache-entry>

<!--
   The LoginCommand and LogoutCommands update a user's account information.
   The cache-entries below invalidate the AccountCommand for an
   individual userID on login and logout
   -->
```

```
    <cache-entry>
      <class>command</class>
      <sharing-policy>not-shared</sharing-policy>
      <name>com.ibm.websphere.samples.trade.command.LoginCommand</name>

      <invalidation>Account_UserID
        <component id="getUserID" type="method">
          <required>true</required>
        </component>
      </invalidation>
    </cache-entry>
    <cache-entry>
      <class>command</class>
      <sharing-policy>not-shared</sharing-policy>
      <name>com.ibm.websphere.samples.trade.command.LogoutCommand</name>

      <invalidation>Account_UserID
        <component id="getUserID" type="method">
          <required>true</required>
        </component>
      </invalidation>
      <invalidation>Holdings_UserID1
        <component id="getUserID" type="method">
          <required>true</required>
        </component>
      </invalidation>

    </cache-entry>

<!--
    The OrderCompletedCommand signifies that a buy or sell
    order has completed for an individual user.
   When an order completes, a user's holdings and account balance
   have been modified.

   This invalidates the AccountCommand, HoldingsCommand and OrdersCommand
   for that user.

   Also, it signifies that an order is completed, therefore,
   on the next request, the ClosedOrderCommand should run to alert the user
   of any completed orders.

    The cache-entries below perform these invalidations for an individual user
    on order completion.
    -->
    <cache-entry>
      <class>command</class>
      <sharing-policy>not-shared</sharing-policy>
      <name>com.ibm.websphere.samples.trade.command.OrderCompletedCommand</name>

      <invalidation>Account_UserID
        <component id="getUserID" type="method">
          <required>true</required>
        </component>
      </invalidation>
      <invalidation>Holdings_UserID
        <component id="getUserID" type="method">
          <required>true</required>
        </component>
      </invalidation>
      <invalidation>Orders_UserID
        <component id="getUserID" type="method">
          <required>true</required>
        </component>
      </invalidation>
      <invalidation>ClosedOrders_UserID
        <component id="getUserID" type="method">
```

```
        <required>true</required>
      </component>
    </invalidation>
  </cache-entry>

<!--
  The Buy and Sell commands initiate a new orderThis creates a new order in the
 database and therefore
  invalidates the OrdersCommand.

  Note that because order-processing in Trade is asynchronous,
  a buy does not invalidate the user's portfolio or
  account balance. These are invalidated when order completes.
  (See OrderCompleteCommand entry in this file.)

  A sell invalidates the user's portfolio
  (to denote that a holding is being sold)
  Buys and sells invalidate the OrdersCommand for the user
  which entered the order.

    -->
    <cache-entry>
     <class>command</class>
     <sharing-policy>not-shared</sharing-policy>
     <name>com.ibm.websphere.samples.trade.command.BuyCommand</name>

    <invalidation>Orders_UserID
      <component id="getUserID" type="method">
        <required>true</required>
      </component>
    </invalidation>
  </cache-entry>

  <cache-entry>
    <class>command</class>
    <sharing-policy>not-shared</sharing-policy>
    <name>com.ibm.websphere.samples.trade.command.SellCommand</name>

    <invalidation>Orders_UserID
      <component id="getUserID" type="method">
        <required>true</required>
      </component>
    </invalidation>
    <invalidation>Holdings_UserID
      <component id="getUserID" type="method">
        <required>true</required>
      </component>
    </invalidation>
  </cache-entry>

<!--
  The UpdateAccountProfileCommand is used to update an individual user's
  account profile information. This invalidates the AccountProfileCommand
  cache-entry above for the individual user, identified by getUserID.
  The getUserID method ties this entry to the dependency id for the
  AccountProfileCommand cache-entry above
  -->
  <cache-entry>
    <class>command</class>
    <sharing-policy>not-shared</sharing-policy>
    <name>com.ibm.websphere.samples.trade.command.UpdateAccountProfileCommand
           </name>
    <invalidation>AccountProfile_UserID
      <component id="getUserID" type="method">
        <required>true</required>
      </component>
    </invalidation>
```

```
   </cache-entry>

<!--
   The ClosedOrdersCommand provides the Order completed alert mechanism in
   Trade. ClosedOrdersCommand is used to check to see if a user
   has any orders which have been closed, but the user has not been
   alerted to the Orders status.

   The ClosedOrdersCommand is cached for each individual user and
   invalidated when an Order is completed by the OrderCompletedCommand
   in this file.
   -->

   <cache-entry>
     <class>command</class>
     <sharing-policy>not-shared</sharing-policy>
     <name>com.ibm.websphere.samples.trade.command.ClosedOrdersCommand</name>
     <cache-id>
       <component type="method" id="getUserID">
         <required>true</required>
       </component>
       <priority>3</priority>
     </cache-id>

     <!-- The dependency id is setup to identify stocks by their symbol
     -->
     <dependency-id>ClosedOrders_UserID
       <component id="getUserID" type="method">
         <required>true</required>
       </component>
     </dependency-id>

     <!-- This dependency id is setup to identify all user Accounts
       for commands which invalidate all Account ClosedOrders cache entries
     -->
     <dependency-id>AllUsers
     </dependency-id>
   </cache-entry>

   <cache-entry>
     <class>command</class>
     <sharing-policy>not-shared</sharing-policy>
     <name>com.ibm.websphere.samples.trade.command.ClosedOrdersCommand</name>

     <invalidation>ClosedOrders_UserID
       <component id="getClosedOrdersCount" type="method">
         <not-value>0</not-value>
         <required>true</required>
       </component>
     </invalidation>
   </cache-entry>


<!--
   The ResetTradeCommand is used to reset the Trade database.
   This action invalidates all cache entries for users and quotes and
   the current MarketSummary
   -->
   <cache-entry>
     <class>command</class>
     <sharing-policy>not-shared</sharing-policy>
     <name>com.ibm.websphere.samples.trade.command.ResetTradeCommand</name>

     <invalidation>AllQuotes
     </invalidation>

     <invalidation>AllUsers
```

```
    </invalidation>

    <invalidation>MarketSummary
    </invalidation>
  </cache-entry>

</cache>
```

Other sources of information for the Tuning WebSphere Application Server Cluster with Caching

Additional resources to provide information on the products, hardware, and software discussed in this study can be found in various books and at various Web sites.

For information about the WebSphere Application Server, see: http://www.ibm.com/software/info1/websphere/index.jsp?tab=products/apptransaction

For earlier information about the WebSphere Application server in a secure environment, see: http://download.boulder.ibm.com/ibmdl/pub/software/dw/linux390/perf/ZSW03003USEN.PDF.

For general instructions on installing and configuring the various components in the topology, see the WebSphere Application Server, Version 6.1 information Center at: http://publib.boulder.ibm.com/infocenter/wasinfo/v6r1/index.jsp

For information about the WebSphere Application Server DynaCache feature, see: http://www.redbooks.ibm.com/redbooks/pdfs/sg247393.pdf

For information about Linux on IBM System z, see: http://www-03.ibm.com/systems/z/os/linux/

For information about z/VM, see: http://www.vm.ibm.com/

For information about the Trade 6.0 application, see the IBM Redbooks Publication "Using WebSphere Extended Deployment V6.0 to Build an On Demand Production Environment" at: http://www.redbooks.ibm.com/abstracts/sg247153.html

For performance information for Trade on the WebSphere Application Server, see: http://www.ibm.com/software/webservers/appserv/was/performance.html

For the caching proxy manuals, see: http://www.ibm.com/software/webservers/appserv/doc/v602/ec/infocenter/index.html

For information about IBM DB2 Database for Linux, see: http://publib.boulder.ibm.com/infocenter/db2luw/v9/index.jsp

For information about IBM open source projects, see: http://www.ibm.com/developerworks/opensource/index.html

Trademarks

IBM Corporation
New Orchard Rd.
Armonk, NY 10504

DB2, DB2 Universal Database, ECKD™, ESCON®, eServer, Express, FICON, HiperSockets, IBM, IBM eserver, the IBM logo, Performance Toolkit for VM, System p, System Storage, Redbooks, System z, System z9, System z10, WebSphere, xSeries, z/OS®, z9, z10, z/VM, and zSeries are trademarks or registered trademarks of International Business Machines Corporation of the United States, other countries or both.

Adobe®, the Adobe logo, PostScript®, and the PostScript logo are either registered trademarks or trademarks of Adobe Systems Incorporated in the United States, and/or other countries.

Cell Broadband Engine™ is a trademark of Sony Computer Entertainment, Inc. in the United States, other countries, or both and is used under license therefrom.

InfiniBand and InfiniBand Trade Association are registered trademarks of the InfiniBand Trade Association.

Java JavaBeans, and all Java-based trademarks are trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.

ITIL® is a registered trademark, and a registered community trademark of the Office of Government Commerce, and is registered in the U.S. Patent and Trademark Office.

IT Infrastructure Library® is a registered trademark of the Central Computer and Telecommunications Agency, which is now part of the Office of Government Commerce.

Intel®, Intel logo, Intel Inside®, Intel Inside logo, Intel Centrino®, Intel Centrino logo, Celeron®, Intel Xeon®, Intel SpeedStep®, Itanium®, and Pentium® are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States and other countries.

LINUX is a registered trademark of Linus Torvalds in the United States, other countries, or both.

Microsoft®, Windows®, Windows NT®, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.

UNIX® is a registered trademark of The Open Group in the United States and other countries.

All statements regarding IBM's future direction and intent are subject to change or withdrawal without notice, and represent goals and objectives only.

Performance is in Internal Throughput Rate (ITR) ratio based on measurements and projections using standard IBM benchmarks in a controlled environment. The actual throughput that any user will experience will vary depending upon considerations such as the amount of multiprogramming in the user's job stream, the I/O configuration, the storage configuration, and the workload processed. Therefore, no assurance can be given that an individual user will achieve throughput improvements equivalent to the performance ratios stated here.

This document is intended to provide information regarding performance of environments using WebSphere Application Server 6.1. It discusses findings based on configurations that were created and tested under laboratory conditions. These findings may not be realized in all customer environments, and implementation in such environments may require additional steps, configurations, and performance analysis. The information herein is provided “AS IS“ with no warranties, express or implied. This information does not constitute a specification or form part of the warranty for any IBM products.