



WebSphere software

IBM WebSphere Application Server for z/OS Version 6: A performance report.

*By Mike Cox, Advanced Technical Support, Americas;
Jim Cunningham, IBM Software Group; Bob St. John,
IBM Systems and Technology Group; Steve Grabarits,
IBM Systems and Technology Group; Linwood Overby,
IBM Software Group; Carl Parris, IBM Systems and
Technology Group; and Gary Puchkoff, IBM Software Group*

Contents

2 Introduction

4 The value of WebSphere Application Server for z/OS and System z

10 Proximity to data: The value of configuring WebSphere Application Server for z/OS in close proximity to operational data source

16 Why run WebSphere Application Server for z/OS?

19 The Mettle Test

24 Business value of the Mettle Test

24 Java performance

27 WebSphere Application Server for z/OS, Version 5.1 performance

28 WebSphere Application Server for z/OS, Version 6.0.1 performance

36 Performance cost of Web-enabling your 3270 application

40 The eRWW workload

44 System z Application Assist Processors

54 Summary

55 For more information

Introduction

The information in this white paper is designed to provide a balance that can suit users of both IBM System z™ mainframes, and Java™ 2 Platform, Enterprise Edition (J2EE) technology-based IBM WebSphere® systems. However, this paper is not designed to be a tutorial on J2EE or a complete articulation of the many unique strengths of the System z server, or the IBM z/OS® operating environment.

When discussing IBM WebSphere Application Server for z/OS performance, it is important to look at performance in the context of an operational environment as opposed to a simple benchmark environment. Using an operational environment enables you to see the true value and strengths of a WebSphere Application Server for z/OS and System z solution. WebSphere Application Server for z/OS is designed to be a cohesive part of the integration platform created by System z servers.

When planning to introduce new workloads into your existing environment, you need help assessing the impact these workloads might have on your existing and future hardware and software systems. Unfortunately, there is no magic formula or set of benchmarks to evaluate system performance. A standardized test is simply unable to provide the specific answers you need for your customized environment. Therefore, the usefulness of any performance data is based on how well you understand your own system and recognize the pieces of benchmark data that are relevant to your business needs.

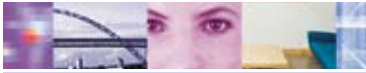
The only accurate method of determining how a server might perform under a particular workload is to test the server within an environment suitable for that workload. Unfortunately, this can be a very difficult and expensive proposition for you to undertake.

To assist you in planning for introducing WebSphere Application Server for z/OS workloads into your existing environments, IBM has documented a combination of primitive component tests, simple end-to-end tests, client proof-of-concept benchmarks and complex operational performance tests. You can choose to use this information to help you make an informed decision, but it is not a substitute for a detailed performance and capacity planning process.

Many organizations are quickly moving their applications toward service oriented architecture (SOA), where back-end transactions are packaged as services with a full suite of common J2EE technology-based connector application programming interfaces (APIs). A variety of popular solution applications, such as those for enterprise resource planning (ERP), collaboration and business intelligence can be accessed through portals and choreographed flows of multiple atomic Web services built with high-level tools into business processes.

Application servers and back-end systems can be both local and remote – accessible by a wide variety of user groups both internal and external. In this environment, the demands on your back-end IT infrastructure become more unpredictable as Web services become exposed to a wider range of applications and a broader audience. The days in which server loads were predictable as the workday progressed through its normal cycle of morning, lunch, afternoon and evening are quickly disappearing as loads become more and more volatile. It is no longer feasible to configure separate server farms for each application and to configure capacity for peak load for each application. And it is also no longer feasible to bring a system down for maintenance when it is so interconnected. Planned and unplanned downtime becomes increasingly disruptive and expensive. The ability to efficiently share compute resources and avoid outages is essential. This capability requires a design focus on operational performance from the ground up – encompassing hardware, firmware, operating systems, transaction managers, databases and other subsystems. IBM has been steadily pursuing this design vision with its System z servers for 40 years. WebSphere Application Server for z/OS is just a more recent step in the evolution of a robust On Demand Business system. It is in this context that this white paper discusses WebSphere Application Server for z/OS performance.

Many different types of workloads and methodologies are used to quantify changes in performance for the comparisons in this white paper. In the lab, primitive measurements are often used to look at the performance of a specific subset of a software stack. These measurements are an important part of the development and performance process to identify changes in behavior and bottlenecks. When primitive measurements improve as a group, the end-to-end measurements tend to reflect these improvements. However, primitive measurements can exaggerate changes in performance. For this reason, it is important to resist the temptation to make a platform judgment based on looking at a single primitive test result. Primitive measurements are not a good indicator of how a system will perform in a complex operational environment. By comparison, you would not make a buying decision between a luxury car or a subcompact car based on spark-plug performance.



Considerations

- Use caution when making decisions about a single data point, particularly with primitive measurements.
- Measurement data is very fluid. It improves frequently. Some data included in this white paper might already be out of date.

The tests discussed in this white paper cover the range of simple primitive tests to complex end-to-end operational-performance tests. The following section briefly describes the different testing categories:

Primitive tests

These tests include:

- Single user, single processor tests, such as *pingMDBQ*
- Multiple-user, multiple-processor tests that drive a single function, such as *pingervlet*
- Tests that have no input or output; data is retrieved from memory, such as primitive Java virtual machine (JVM) measurements

End-to-end tests

Multiple users drive multiple transactions through the WebSphere Application Server for z/OS run time to a back-end database (for example, Trade6 or e-business Relational Warehouse Workload [eRWW]).

Client benchmarks

These tests are usually simpler than production applications. However, they exercise the basic design and logic of the application and run against real client data.

Operational-performance tests

These tests consist of mixed workloads, multiple application priorities, multiple partitions, load spikes, failure recovery and so on. They measure the system's ability to respond to operational challenges with acceptable performance. An example of this kind of test would be the Mettle Test. You can view the results of the Mettle Test at ibm.com/software/webservers/appserv/zos_os390/mettle.html.

The value of WebSphere Application Server for z/OS and System z

WebSphere Application Server for z/OS, Version 6 has greatly evolved from Version 4. Version 6 features a common programming model and is built from common code. Version 4 featured distributed IBM WebSphere Application Server products, as well as a totally unique WebSphere Application Server for z/OS product. WebSphere Application Server for z/OS, Version 4 had a different programming model, different APIs, interfaces, application packaging requirements, among others. WebSphere Application Server for z/OS, Version 6 enables administration of its nodes on varying platforms and releases that can be contained within a cell.

The WebSphere Application Server for z/OS product has evolved into a mature, strategic key piece of software for the mainframe. As SOA is becoming more prevalent, core mainframe assets are more commonly packaged as services and reused with new J2EE applications. This makes the unique platform abilities of WebSphere Application Server for z/OS more attractive than ever.

State-of-the-art transactional run time

When discussing the performance of WebSphere Application Server for z/OS, you must first look at the how the WebSphere Application Server for z/OS runtime structure has been designed. It differs from the other J2EE run times in some very important ways that relate to achieving operational performance (see Figure 1).

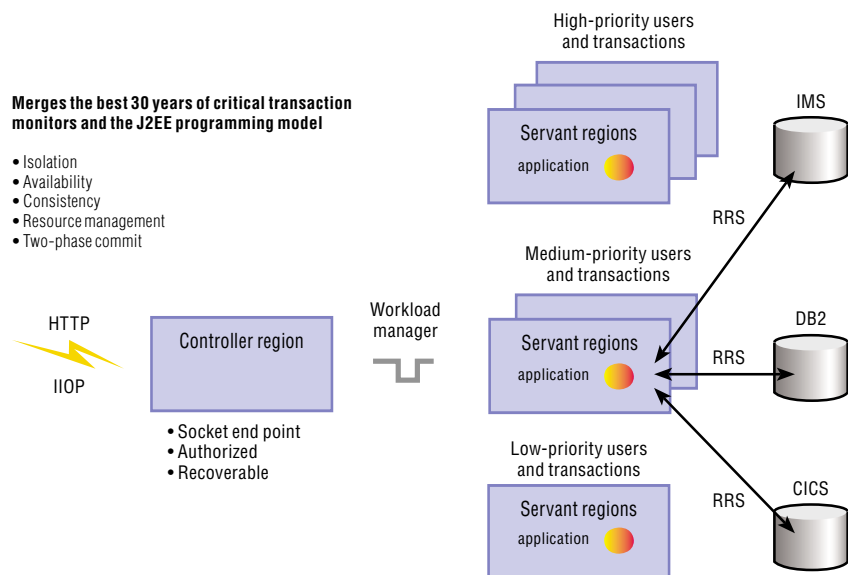


Figure 1. State-of-the-art transactional run time

As you move in Figure 1 from left to right, you see HTTP, or Internet Inter-ORB Protocol (IIOIP) requests coming in from the network (or local clients) to a z/OS address space called a *controller region*. The controller region is an authorized, recoverable address space that listens on a TCP/IP socket and then routes incoming work to one or more address spaces called *servant regions*. These servant regions implement the J2EE artifacts associated with the work of the transaction, such as servlets, JavaServer Pages (JSP), session Enterprise JavaBeans (EJB), bean-management persistence (BMP) and container-managed persistence (CMP) artifacts and message-driven beans (MDBs). Thus, a single WebSphere Application Server for z/OS runtime instance, or server, is actually made up of multiple address spaces.¹ The multi-address space design provides a number of advantages.

¹ There are additional address spaces that are part of the run time, which are used for startup and administrative functions that are not part of the mainline performance path. These address spaces will not be discussed in this white paper.

Workload management

When work is distributed to a servant region from the controller region, it is placed on a z/OS Workload Manager (WLM) queue. The z/OS WLM classifies the unit of work by a number of criteria, including server name, server instance name, user ID and transaction class. Transaction classes can be assigned using a Workload Classification XML document, which enables HTTP requests to be classified by host, port or Uniform Resource Identifier (URI); IIO requests to be classified by application, module, component or method; and MDBs to be classified by message listener port and selector attribute. An installation using WLM defines one or more service classes according to service-level agreement (SLA) criteria, such as *90 percent of the transactions in a service class must be completed in less than half a second*. When multiple service classes are defined, some can be designated as more important than others according to importance assignments. WebSphere Application Server for z/OS, interacting with WLM, monitors the performance of each service class to determine if the work in that class is meeting its SLA criteria. If work is falling behind or a spike in the load occurs, WLM can start another servant region, adding resources to handle the additional workload, which in turn improves performance. If the system is constrained, WLM can shift resources from lower-priority work to support higher-priority work.

WLM is designed to be in the mainline path of the run time. It consistently monitors the behavior of transactions running in each service class and providing a tight feedback loop deeply integrated into the z/OS system. This capability provides real-time adjustments to system components to help maximize the processing resources to help ensure that higher-priority work is serviced before lower-priority work.

WLM also provides another advantage for the z/OS installation. Report classes can be assigned to different work based on all the criteria listed previously. This capability enables the installation to easily isolate the resource requirements and responsiveness of each category of work.

Address-space isolation

When using multiple servant-region address spaces, a defective transaction that causes the address space to fail only affects the transactions running within that servant region. This capability provides a high degree of application isolation and positively affects availability. If an address space does fail, WLM detects the loss of resources and immediately starts a new servant region and begins routing work to it.

Global transactions and two-phase commit

When running transactions with a global commit scope (with TRANSACTION_REQUIRED as the default), WebSphere Application Server for z/OS registers the commit scope with Resource Recovery Services (RRS). RRS provides operating-system control of the commit process. When performing a full two-phase commit with more than one resource manager such as IBM IMS™, IBM CICS®, IBM DB2® and IBM WebSphere MQ, RRS can handle the full two-phase-commit coordination process with highly optimized internal z/OS code. This capability has several advantages:

- *Performance is improved because the commit process is internal to the operating system rather than in middleware. When you are performing a two-phase-commit interaction, locks are held on more than one back-end data resource. The longer these locks are held, the greater the negative impact on performance. By optimizing this path, you can enable locks to be released sooner—and help benefit all work.*
- *Not all resource adapters can handle full two-phase-commit interactions with Extended Architecture (XA) middleware without some compromise in commit state.*
- *When a z/OS system handles two-phase commit, it helps simplify and even eliminate application management and coding of rollback logic.*

The previously mentioned tightly integrated functions of WLM, failure isolation and two-phase commit are all transparent to the J2EE application and thus, the application developer. This design offers the benefits of a cluster without actually configuring a cluster using WLM-based routing and integrated high-availability features. Any J2EE technology-compliant enterprise archive (EAR) file can deploy transparently in the WebSphere Application Server for z/OS run time without your needing to modify the application – and inheriting many of the built-in qualities of service of the System z platform.

Since Version 4, WebSphere Application Server for z/OS has been fully J2EE compatible. However, WebSphere Application Server for z/OS differs from other distributed J2EE run times in the industry. WebSphere Application Server for z/OS takes full advantage of the strengths of the z/OS and System z platform. IBM has incorporated into WebSphere Application Server for z/OS the lessons learned from more than 30 years of experience designing and optimizing critical transaction managers in complex operational environments. These capabilities in other J2EE run times are managed externally to the operating system through middleware add-ons that increase the level of complexity and can result in less system integration and additional path length.

The capabilities discussed previously are not demonstrated in many of the primitive and single instance end-to-end benchmarks mentioned in this white paper. However, performance tests for a variety of workloads have consistently shown high *n*-way scalability with symmetric multiprocessing (SMP) ratios higher than traditional existing Large Systems Performance Reference (LSPR) workloads using the basic design in WebSphere Application Server for z/OS, Version 4, Version 5 and now Version 6. The description of the Mettle Test on page 19 illustrates the operational performance value of these features in a dynamic multipartitioned environment running a mix of high- and low-priority workloads.

WebSphere Application Server for z/OS high-availability client POC configuration

Shortly after the general availability of WebSphere Application Server for z/OS, Version 5, a large financial institution tested its proof-of-concept (POC) online bank-teller application in the IBM Montpellier Systems Center (see Figure 2). The bank required extreme scalability, subsecond response time and high availability. Its application represented a mix of automatic teller machine-like transactions. The application consisted of servlets driving Java Database Connectivity (JDBC) calls to IBM DB2 Universal Database™ on z/OS with an average of five to six Structured Query Language (SQL) calls per transaction.

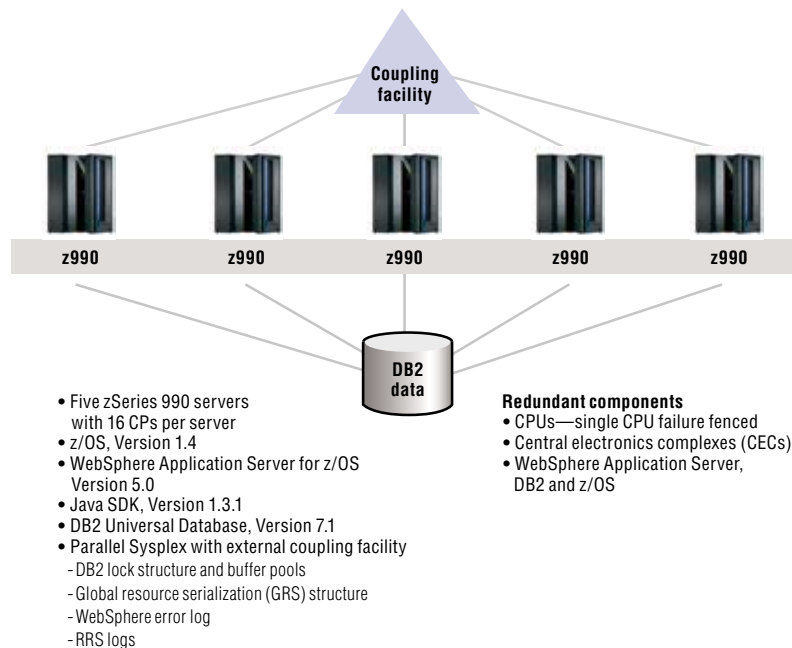


Figure 2. WebSphere Application Server for z/OS high-availability client POC configuration

As shown in Figure 2, five 16-way IBM @server® zSeries 990 (z990) systems were configured as a single-system image within an IBM Parallel Sysplex® environment. A single DB2 database was shared by all the systems in the cluster. A coupling facility consisting of four processors was used for DB2 buffer pools, the DB2 lock structure and IBM MVS™ functions required to support a sysplex. WebSphere Application Server for z/OS is designed to be easily configured into a Parallel Sysplex configuration and can use the coupling facility for a number of functions. In this case, the coupling facility contains WebSphere Application Server for z/OS error logs and the RRS log.² Each z990 box was configured with WebSphere Application Server for z/OS, Version 5, DB2 Universal Database, Version 7.1 and the z/OS, Version 1.4 operating system.

In addition to configuring a Parallel Sysplex environment for high-end scalability, you can use a single-system image Parallel Sysplex configuration to provide high availability with full redundancy in software and hardware. Duplexing³ is often used in production environments, but in this example, duplexing was not applied to the coupling facility.

WebSphere Application Server for z/OS sysplex client POC results

Figure 3 illustrates the measured results.

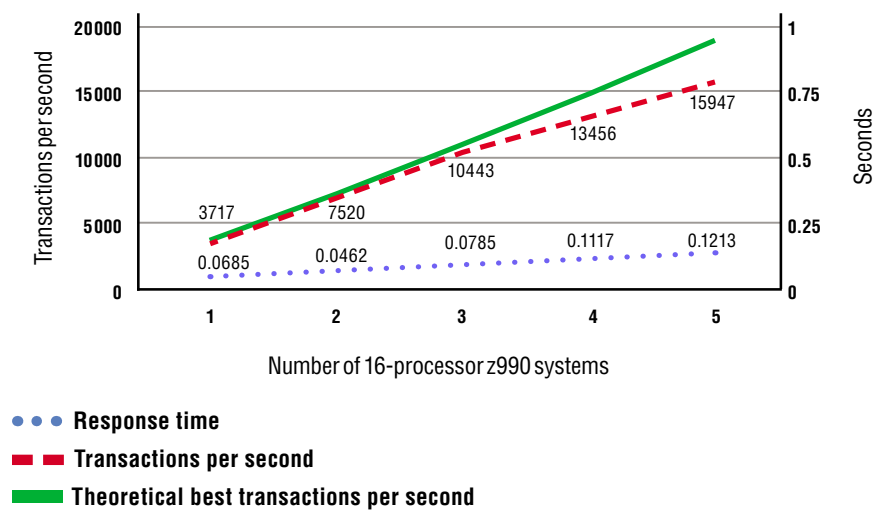


Figure 3. WebSphere Application Server for z/OS sysplex client POC results

² WebSphere Application Server for z/OS can also use the coupling-facility structure for shared persistent session data and Security IDs (SIDs).

³ System-managed coupling-facility structure duplexing is designed to provide a general-purpose, hardware-assisted, easy-to-use mechanism for duplexing coupling-facility structure data. It provides a robust recovery mechanism for failures, such as loss of a single structure or coupling facility, or loss of connectivity to a single coupling facility, through rapid failover to the other structure instance of the duplex pair.

In Figure 3, the y axis on the left represents the throughput for the configuration, measured in transactions per second. The y axis on the right represents response time as measured by the external load emulator. The x axis shows the number of 16-processor z990 boxes. The dashed line graphs the measured transaction rate as the number of boxes configured in the sysplex was scaled up. The solid line shows the theoretically best scaling potential based on the throughput of one z990 box. The measured results scaled extremely well considering that the DB2 data is shared and the SQL activities include updates. Also, these scaling numbers were achieved while consistently providing a less than one-quarter-second response time on a system running at 95 percent utilization as shown by the dotted line.

These results were obtained using WebSphere Application Server for z/OS, Version 5 just after it became generally available. Improvements discussed in this document for WebSphere Application Server for z/OS, Version 5.1 and WebSphere Application Server for z/OS, Version 6 have the potential to improve these transaction rates.

Proximity to data: The value of configuring WebSphere Application Server for z/OS in close proximity to operational data sources⁴

The J2EE programming model is a distributed object model that provides flexibility in application design, as well as a multitude of deployment options. However, you should consider some important performance factors when making deployment decisions. WebSphere Application Server for z/OS offers opportunities for local optimizations when deployed locally with back-end DB2 and transaction managers. The following two examples demonstrate measured performance improvements realized by clients when they redeployed or refactored their applications in close proximity to the data that the applications accessed. These were controlled benchmarks using the client's own applications. First, consider a basic J2EE flow as shown in Figure 4.

⁴ The discussion in this section is based on a Washington Systems Center technical document entitled, *Optimizing WebSphere for z/OS Performance* written by Mike Cox and Paul Glass. For a more detailed discussion of this topic, you can read this article at ibm.com/support/techdocs/atmastr.nsf/WebIndex/WP100558.

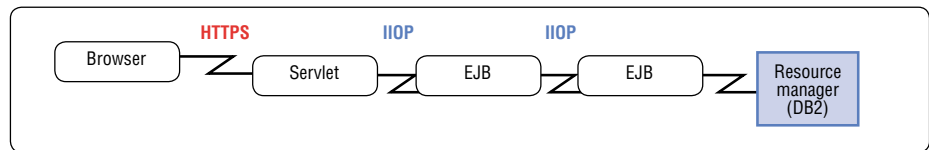


Figure 4. A typical J2EE flow

In this example, the TCP/IP flows can be in a J2EE request: client to Web container, Web container to EJB container, EJB container to EJB container and EJB container to back-end database (DB2 in this example). Some of these TCP/IP flows are Remote Method Invocation over Internet InterORB Protocol (RMI/IIOP) calls between the J2EE components. RMI/IIOP processing can be expensive because it includes serialization (making a copy of the data that is formatted to be passed over TCP/IP) and deserialization (essentially rebuilding serialized objects into full Java objects). If all the EJBs required for an application can be found in the same EJB container, you can avoid serialization and deserialization cost. Local EJB interfaces can be defined, or the pass-by reference⁵ option can be used so that objects passed as parameters from one EJB to another do not need to go through this costly processing for each EJB call.

In the next section, you can see the benefits of reducing the number of remote EJB calls per transaction, which can help reduce serialization cost and network latency. The second example demonstrates the potential savings associated with making local JDBC calls using the Type-2 JDBC option as opposed to a remote Type 4-JDBC call.

⁵ *Pass by reference* is a parameter that you can select through the WebSphere Application Server for z/OS administrative console when applicable.

Proximity to data: Transportation industry benchmark—refactoring an application to avoid remote EJB-to-EJB interactions

The following benchmark comparison was performed at IBM Washington Systems Center using a benchmark provided by a large transportation company for its customer information system. In Figure 5 the configuration shows the base case with the original configuration tested. The application flow consisted of a Web container and an EJB container running in WebSphere Application Server for z/OS on an IBM System p™ machine with the IBM AIX® operating system. The application servlet running in the Web container made a local IIOp call to a session EJB that contained the business logic of the transaction. The session EJB then made several RMI/IIOp calls to CMPs running in WebSphere Application Server for z/OS, which then accessed DB2 using Type-2 JDBC calls. In this case, DB2 is configured as a Parallel Sysplex environment for both scalability and availability.

The business-logic session EJB was running remotely to the CMP EJBs, which ran the SQL calls to DB2. To make matters worse, for each transaction, the business-logic EJB had to make several calls to these remote CMP EJBs. As a result, these RMI/IIOp calls added significant serialization and deserialization overhead.

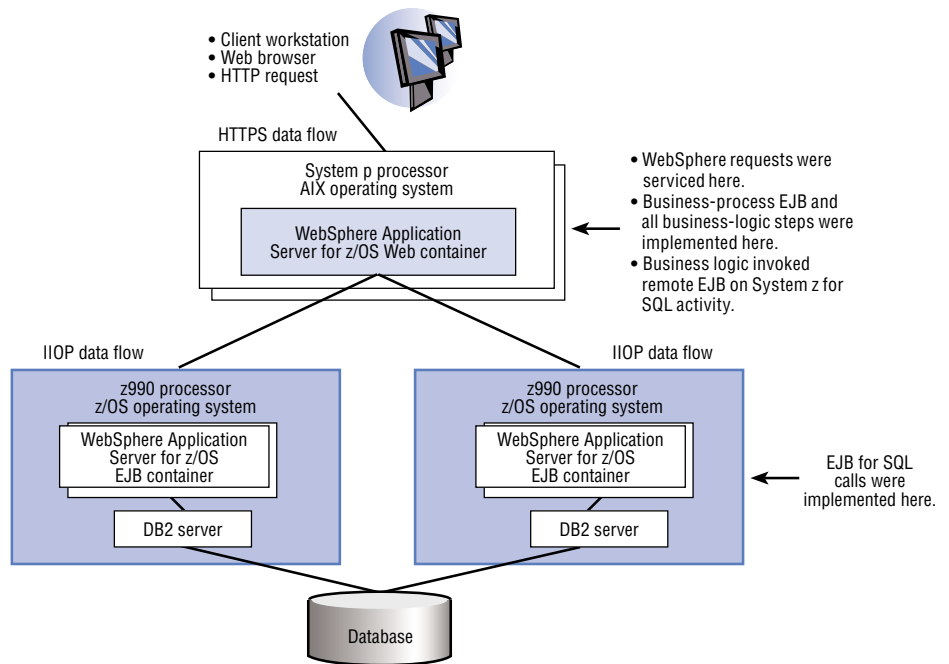


Figure 5. Remote (distributed) business-processing logic environment

Figure 6 shows the same physical configuration, but in this case, the application has been refactored to place the business-logic processing in the same EJB container with the data access CMP EJBs in the WebSphere Application Server for z/OS run time. Doing this helped significantly reduce the traffic from the System p machine to the System z machines. In addition, local interfaces in the application were defined for the EJBs that shared the EJB container on z/OS. Doing this helped reduce the number of remote EJB calls per transaction, helping to significantly reduce the serialization and deserialization cost.

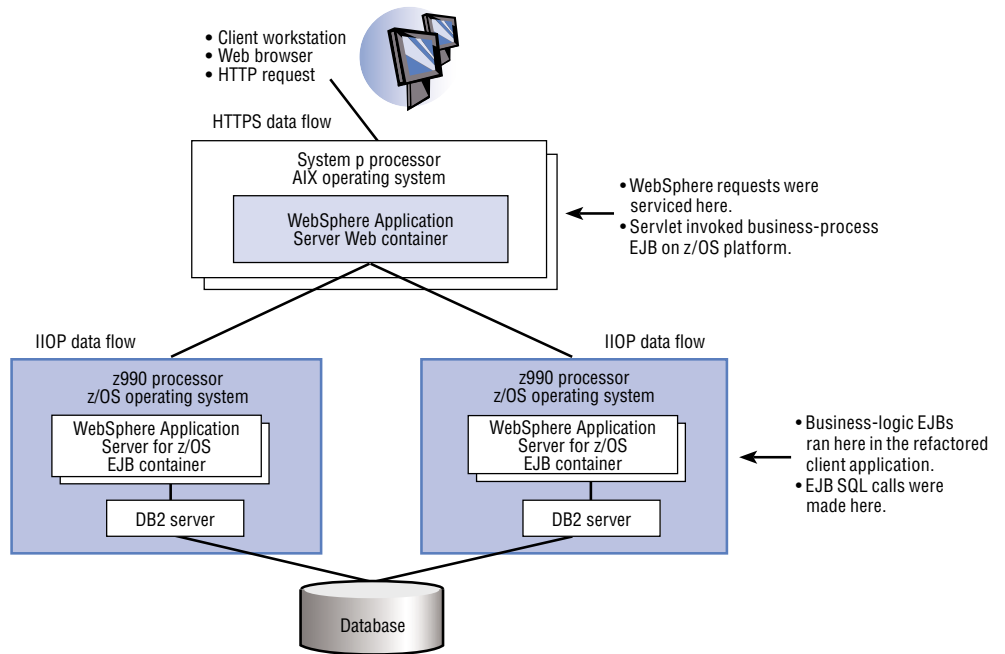


Figure 6. Local z/OS business-processing logic environment

Table 1 shows that there was a very significant difference in the overall performance of these two application-deployment approaches.

- Average processor time per EJB transaction was reduced by more than 77 percent.
- The data volume per EJB transaction was reduced by 99 percent.

Benchmark configuration	Average processor time per EJB transaction (ms)	Amount of data transferred per EJB transaction (KB)
Remote (distributed) business-logic environment	11.73	54.4
Local z/OS business-logic environment	2.64	0.5

Table 1. Comparison of local and remote IIOp performance

By putting both the business and data logic in the same WebSphere Application Server for z/OS EJB container and minimizing the interactions between the Web tier and the EJB tier, fewer cycles were spent in serialization and deserialization, and less data was transferred over TCP/IP, which helped result in reduced processor consumption, bandwidth and latency.

Proximity to data: Finance industry benchmark—reducing physical tiers to optimize local access to DB2

The following benchmark comparison was performed at a client site using the client’s configuration with the assistance of IBM expertise. In this case, the client was running a multitier configuration illustrated in Figure 7.

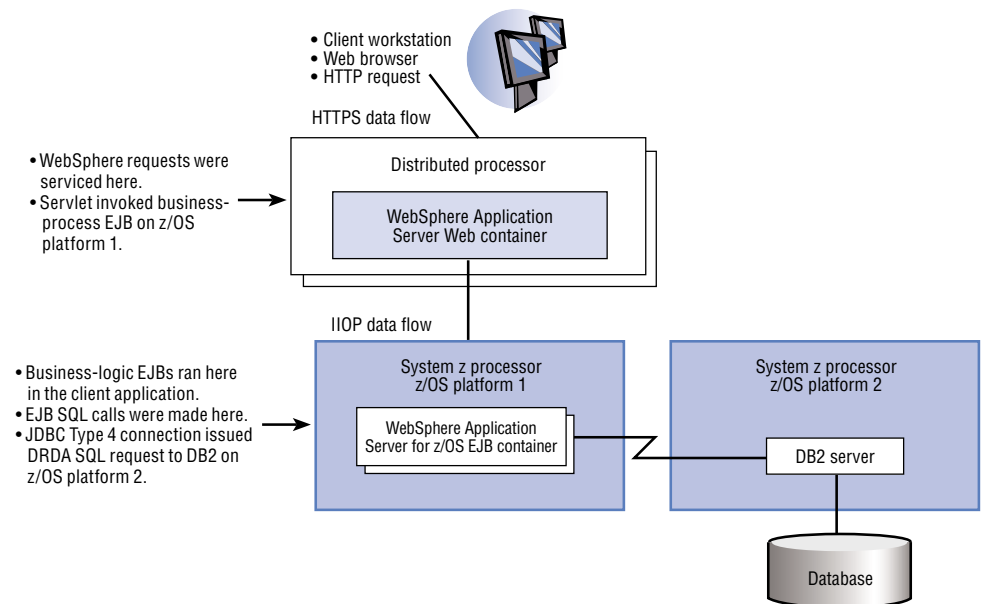


Figure 7. Financial client with remote (distributed) enterprise database

In this example, the first step is a WebSphere Application Server for z/OS container operating on a distributed System z box. The distributed box is running a servlet that makes a remote IIOOP call to WebSphere Application Server for z/OS on System z, which is running the session EJB business logic. Next, a local IIOOP call is made to the data-access CMP EJB, which then makes SQL calls to DB2 on z/OS. DB2 on z/OS is running on another System z box. The CMP EJB SQL calls to DB2 on z/OS are using a remote Type-4 JDBC connection which goes through Distributed Relational Database Architecture (DRDA). This configuration is running WebSphere Application Server for z/OS as a distributed tier, not taking advantage of the potential for closer proximity to the Web container or closer proximity to the back-end data.⁶

⁶ A number of factors contributed to the separation of the Web container from the EJB container in this case, which were unrelated to performance considerations.

When accessing DB2 remotely, a number of steps take place. In the case of DB2 Universal Database for z/OS, a Type-2 JDBC driver can be used when the J2EE components and DB2 subsystem reside on the same operating-system image. When the Type-2 JDBC driver is used, data can be passed to DB2 in a more-usable format than a Type-4 connection, and the TCP/IP flows are eliminated. When using the Type-2 JDBC driver, tasks are not redispached, and the DB2 activity continues on the current implementation thread. The WLM priority of the current request being processed is maintained. In addition, the Type-2 JDBC driver uses RRS to manage transaction state, which performs better than the Type-4 XA JDBC provider.

Figure 8 shows the configuration when it has been changed to take advantage of local Type-2 JDBC access to DB2.

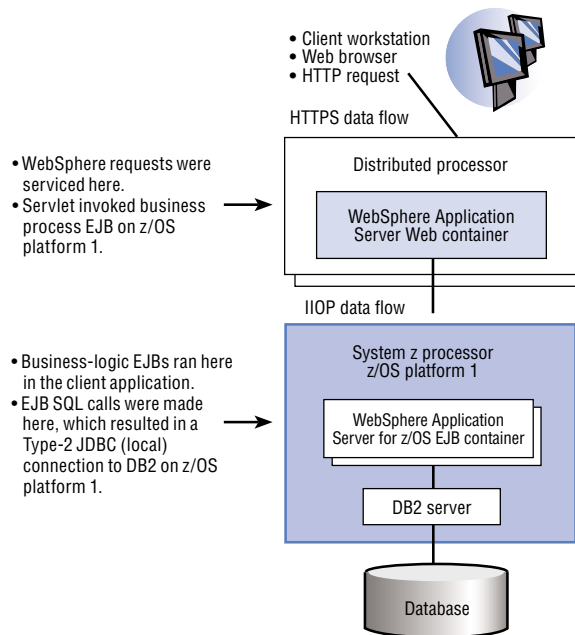


Figure 8. Financial client with local enterprise database

In the above case, the physical configuration has been modified to eliminate one of the tiers in the flow. The business-logic session EJB still makes a local call to the CMP EJB, which runs SQL calls to a local instance of DB2 Universal Database on z/OS using the local Type-2 JDBC connection. Changing from a Type-4 to Type-2 JDBC connector is a simple process of defining a new data source and configuring the application to use it.



Why run WebSphere Application Server for z/OS?

Integrated sysplex functionality that helps enable you to:

- Scale using the sysplex, while transparently supporting client-affinity redirection for state management
- Dynamically upgrade to new operating-system and database releases without bringing the system down
- Survive a software subsystem outage without losing the availability of the entire system

IBM System z hardware that enables you to:

- Survive a processor failure without losing the availability of the entire system
- Dynamically add capacity to respond to unexpected workload spikes

The ability, since Version 4, of WebSphere Application Server for z/OS and RRS to support application models that require two-phase commit across IMS, CICS and DB2

State-of-the-art resource management that enables you to differentiate and prioritize work based on service-level requirements, using WLM in WebSphere Application Server, WLM in z/OS, IBM Intelligent Resource Director and WLM with a sysplex distributor.

Deep, end-to-end security integration that provides:

- Enhanced IBM RACF® controls for user access to the TCP/IP stack, ports and network
- Integrated intrusion-detection services for port scanning, stack attacks and flooding detection

Significant improvements in performance were observed as a result of this change. A 50 percent reduction in average (Web) user response times were reported by the test driver tools. Also, the overall processor power required by the z/OS system environment was 50 percent less than the remote (to z/OS) system implementation. In essence, moving to a Type-2 driver resulted in performance improvements and less processor churn.

Proximity to data: Summary

The client-based benchmark examples demonstrate the real and potential performance advantages provided by consolidating J2EE business logic and enterprise data access on a single z/OS tier. Reduced processor consumption, improved response times and reduced bandwidth requirements combine with other quality-of-service advantages, such as resource management, transaction-commit scope management and security to make WebSphere Application Server for z/OS an attractive option for critical J2EE technology-enabled applications.

Why run WebSphere Application Server for z/OS?

WebSphere Application Server runs on many platforms. It is important to understand why you should choose to run WebSphere Application Server on z/OS. As discussed previously in this white paper, the WebSphere Application Server for z/OS run time has a unique design that takes advantage of many z/OS and System z features to provide industry-leading qualities of service, such as the following capabilities:

Integrated sysplex functionality

WebSphere Application Server for z/OS is designed to easily take advantage of Parallel Sysplex functionality. Along with the well-known advantages of scalability (as demonstrated previously in this white paper) and availability, WebSphere Application Server for z/OS can use the coupling facility for error and RRS logging, persistent session durability, client-affinity redirection for state management, and a security ID cache. In the case of a WebSphere Application Server for z/OS failure, the previously discussed data can remain available to other WebSphere Application Server for z/OS images configured in the sysplex. WebSphere Application Server for z/OS maintenance can be applied without affecting availability. The sysplex distributor interacting with WLM dynamically routes incoming WebSphere Application Server for z/OS requests to the least-busy WebSphere Application Server for z/OS instance.

System z hardware

System z servers provide near-continuous reliable operation (CRO) and deliver high-availability solutions. High-availability solutions yield the industry's highest mean time between failure (MTBF) results for unscheduled system outages. The z990 family of processors are designed to have a MTBF of more than 30 years, with recent field performance exceeding the design target based on current worldwide client field data. If a single processor fails, a spare processor can be switched in automatically by the firmware. Even in the rare case where no spare processing units are available, then the failed processor is fenced off from the system, and in the vast majority of cases, doesn't affect any running applications.⁷

Integrated two-phase commit

The z/OS operating system handles two-phase-commit processing using the tightly integrated RRS function. Unlike distributed XA function, RRS is not managed externally to the operating system in middleware. Providing two-phase-commit processing within the operating-system layer provides performance advantages by helping to reduce lock-held time in each of the back-end enterprise information systems (EISs) within the commit scope, while also helping to improve the performance of single-phase-commit transactions running concurrently against the same resources. RRS supports local two-phase-commit processing between WebSphere Application Server for z/OS, IMS, CICS, WebSphere MQ for z/OS and DB2 Universal Database for z/OS.

Because the RRS component of the z/OS operating system handles global commit rollback, you don't have to provide additional compensation or rollback logic in the business logic of the application, thus helping to simplify application development.

State-of-the-art resource management

Four mechanisms are provided to differentiate between different priorities of work on the system using client-defined SLA criteria through WLM.

- *WLM within the WebSphere Application Server for z/OS run time manages priorities between different WebSphere Application Server for z/OS transactions.*
- *WLM within z/OS manages priorities between WebSphere Application Server for z/OS and non-WebSphere Application Server for z/OS applications (such as Java batch applications and existing applications) running within the same z/OS instance.*

⁷ For more information about CRO and high-availability solutions, visit: ibm.com/journal/rd/435/mueller.html.

The reliability, availability and serviceability (RAS) strategy for z990 and IBM System z9™ mainframes is to continue the IBM S/390® objective of providing CRO. This RAS strategy is constructed with a set of building blocks that work closely together: error prevention, error detection, error recovery, problem determination, service structure, change management, and RAS measurement and analysis. The interdependency among the building blocks is such that removing or weakening any of them limits the ability of the design to achieve the overall CRO objective. Each building block must be fully implemented and must run flawlessly within itself and together with the other blocks.

- *WLM of all service classes running across a multipartitioned configuration using Intelligent Resource Director enables the shifting of resources to high-priority WebSphere Application Server for z/OS work in one partition from lower-priority work in another partition. This capability uses the Parallel Sysplex coupling facility and can monitor the work running in all of the logical partitions (LPARs) of a multibox sysplex.*
- *WLM, in conjunction with the sysplex distributor, intelligently routes incoming WebSphere Application Server for z/OS requests to the least busy WebSphere Application Server for z/OS partition when multiple WebSphere Application Server for z/OS partitions are configured. Although the top three examples of WLM usage move resources to the work, sysplex distributor routes work to the resources.*

No other system in the industry has this level of resource-management integration with hooks in the hardware, hypervisor, operating system and the J2EE run time.

Deep end-to-end security integration

WebSphere Application Server for z/OS is designed with a System Authorization Facility (SAF) interface. You can continue to use your existing security definitions in your security subsystem when running WebSphere Application Server for z/OS. Tight integration with the security component means that authentication, authorization and auditing can all be performed using the same administration on z/OS as traditional z/OS subsystems. Also, taking advantage of z/OS cryptographic hardware means strong, fast, and security-rich processing for things like Secure Sockets Layer (SSL) connections.

Network access to WebSphere Application Server for z/OS is provided by the z/OS communications server TCP/IP. The communications server maintains a hardened line of defense for the z/OS system and provides a highly secure network infrastructure on which to deploy WebSphere Application Server. The communications server uses SAF protection through the SERVAUTH class to help ensure that local users have permission to access TCP/IP resource networks, such as population of fast response cache accelerator (FRCA) Web content, access to local ports, network resources and even the TCP/IP stack itself. The communications server also has a built-in intrusion detection system (IDS) that reports and defends against attacks on the z/OS network layers. The IDS is policy-based, rather than signature-based, and uses its position as the server end point to detect both known and unknown attacks that might otherwise go undetected by outboard network-based intrusion-detection devices. The IDS detects network and port scans, single-packet attacks, and multipacket denial-of-service attacks.

The Mettle Test

Most of the performance data referenced in this white paper has been measured running in a controlled lab environment with care taken to help ensure that there haven't been any configuration bottlenecks, such as lack of memory, communications adapters and disks. The measurements were usually taken during a sample interval where the workload had achieved a steady state. Tests were usually done in dedicated partitions with nothing else running on the system. Although this is valuable for performance analysis to highlight changes in various components, it is not the way an enterprise might run in real-world operational environments.

Also, because of the previously discussed lab-evaluation methodology, many of the value-added functions of the WebSphere Application Server for z/OS run time are not required or demonstrated. In fact, having WLM, RRS, security and cross-address-space communications in the mainline paths adds some processor costs to the transactions in spite of the attention paid to optimize these functions.

The WebSphere Application Server for z/OS on System z development teams have made a conscious decision to optimize for operational performance rather than benchmark performance. Ultimately, in the real world, this test has the ability to deliver more-usable performance than is demonstrated in benchmarks that run a single workload at steady state on a single operating system on a single box with nothing else running in the configuration. To fully optimize all of the resources on a system while running multiple workloads of different priorities, it is necessary to take a system view of performance and to design the qualities of service into the hardware, firmware, operating system and WebSphere Application Server for z/OS run time.

The Mettle Test is an operational performance demonstration of the unique capabilities of the WebSphere Application Server for z/OS running on a System z machine.⁸ The Washington Systems Center configured the system in Figure 9 for this test.

⁸ To see the full demonstration, download the Mettle Test Flash demonstration from ibm.com/software/webservers/appserv/zos_os390/mettle.html. The information in this white paper only touches the surface. The demonstration package includes in-depth descriptions of WebSphere Application Server, z/OS and System z functions, such as WLM, Intelligent Resource Director, Parallel Sysplex, hardware and software recovery, and maintenance, along with screen images taken from a running system.

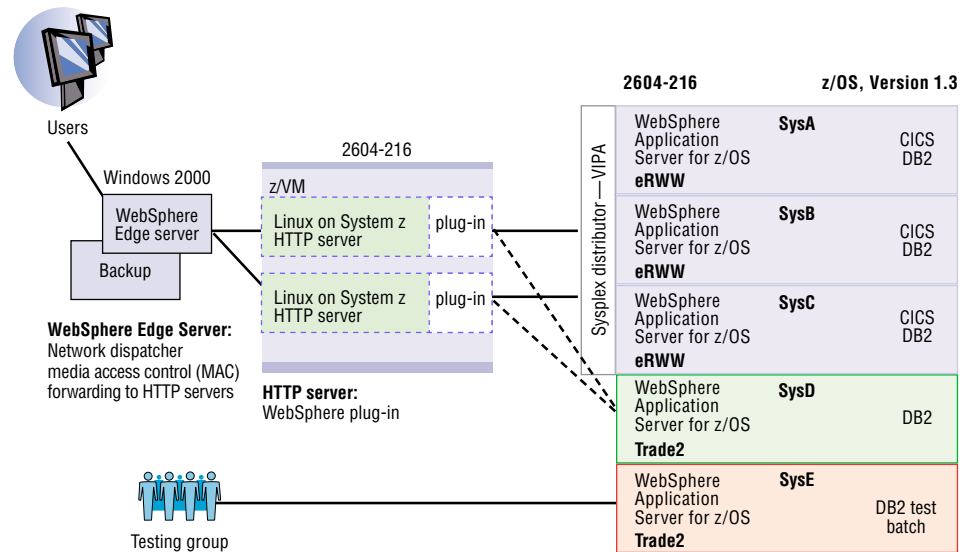


Figure 9. The Mettle Test configuration

The configuration was built with high-availability considerations. Incoming client requests drove a pair of IBM WebSphere Edge Server systems (one functioned as back-up) running on Microsoft® Windows® 2000.⁹ The edge servers drove a pair of HTTP Web servers and plug-ins, each running in a Linux® environment on a System z guest on the IBM z/VM® platform. The HTTP servers passed transactions to WebSphere Application Server for z/OS running in four LPARs on a z900 16-processor system. These four LPARs (SysA, SysB, SysC and SysD) ran “production” applications. A fifth partition (SysE) simulated a development and test partition running a WebSphere Application Server for z/OS workload and a non-WebSphere Application Server DB2 batch workload. The five LPARs shared the 16 physical processors configured on the z900 machine.

SysA, SysB and SysC were configured as a single system image in a Parallel Sysplex environment, sharing a single copy of the DB2 data. These partitions ran an order-inventory workload called *eRWW* (see page 40), which consists of WebSphere Application Server for z/OS driving the IBM CICS Transaction Gateway J2EE Connector Architecture (JCA) to CICS Transaction Server and DB2. Eight transactions are in the workload ranging from high-volume, trivial transactions, to medium to heavy queries and a processing-intensive query. All but the processing-intensive query are the highest-priority transactions in the system. The processing-intensive transaction is low priority. The workload has an 80/20 read-write ratio. HTTP requests to SysA, SysB and SysC used the sysplex distributor and virtual IP addressing (VIPA) to route requests to the least-busy WebSphere Application Server for z/OS instance based on dynamically monitored WLM statistics.

⁹ At the time of the Mettle Test, WebSphere Edge Server was not available on Linux on System z. It now runs on this operating environment and could be configured in Linux on System z guests.

SysD ran the Trade2-EJB benchmark in “production.” SysE ran Trade2-EJB as a “test” workload along with DB2 batch. Each of the workloads in the system had SLA criteria and were assigned to service classes with a range of priorities.

This benchmark system configuration was confronted with a series of 10 operational challenges to demonstrate operator-free resource management according to SLA specifications, as well as unplanned- and planned-outage avoidance. Variations in high- and low-priority loads and numerous failures were tested. System behavior was monitored on custom gauges designed specifically for the Mettle Test.

The goals of the dynamic resource management (self-optimizing) scenarios included:

- *Distinguish between high- and low-priority WebSphere Application Server for z/OS users*
- *Distinguish between high- and low-priority applications*
- *Respond with flexibility to changing processing capacity requirements with Intelligent Resource Director*

The goals of the planned- and unplanned-outage avoidance (self-healing) scenarios included:

- *Isolate WebSphere Application Server for z/OS application failures*
- *Provide continuous availability of the sysplex*
- *Isolate and recover hardware failures*
- *Provide nondisruptive installation of application and middleware maintenance*

Self-optimization tests

Self-optimization refers to the ability of systems or components to efficiently maximize resource allocation and usage to meet user needs without human intervention. Typically, self-optimization addresses the complexity of managing system performance and workload balancing. More-advanced self-optimizing components, such as those found in System z servers and the z/OS operating system, are designed to learn from experience, and automatically and proactively tune themselves in the context of overall service for both system users and their customers.

The Mettle Test included two self-optimization scenarios:

- *Managing goals in a system*

The first self-optimizing scenario demonstrated that z/OS WLM can distinguish between high-priority order-entry WebSphere Application Server for z/OS work and low-priority data-mining work running in the same WebSphere Application Server for z/OS image, and autonomically adjust system resources to help ensure that the higher-priority work is completed within its stated business objectives.

- *Managing goals across systems*

The second self-optimizing scenario demonstrated that System z Intelligent Resource Director can extend this autonomic workload management beyond a single system by redirecting resources from a lower-priority z/OS image (in this case, a z/OS system running test and batch) to a higher-priority z/OS image running the WebSphere Application Server for z/OS workload.

Because z/OS is a system that exhibits self-optimizing behavior, and WebSphere Application Server for z/OS is tightly integrated with the self-optimizing algorithms of WLM, the resources of each z/OS image of the Mettle Test were properly and automatically assigned so that the critical, higher-priority work wouldn't miss SLA criteria. The test also showed WLM clamping down on the resources available to the lower-priority work until the higher-priority work was done. When the resources were freed, they were automatically reassigned to the lower-priority data-mining work.

Self-healing tests

Self-healing systems can detect improper operations – either proactively through predictions, or reactively – and initiate corrective action without disrupting system applications. Corrective action could mean that a product alters its own state or influences changes in other elements in the environment. Having self-healing capabilities helps the system to ensure that day-to-day operations do not falter or fail because of events at the component level. Likewise, the system as a whole becomes more resilient as changes in the system are made to reduce or eliminate the business impact of failing components.



The Mettle Test scenarios graphically illustrate what happens when an important application experiences a sudden spike in activity when the system is already 100 percent busy, and when an LPAR running the critical work in a sysplex fails. The scenarios also show what happens when a processor fails with six different workloads running across five LPARs sharing 16 physical processors.

The Mettle Test included several self-healing scenarios:

- *Recovering from application failure*
In this scenario, a poorly written application with a memory leak was injected into a stable, running system. Because of the advanced architectural structure of WebSphere Application Server for z/OS, the failure was isolated and fenced off, and the system structures were repaired without any loss of availability.
- *Recovering from system failure*
One of the operating-system images running the critical order-entry workload was abnormally ended. In response, z/OS subsystems automatically restarted on the remaining systems to clean up resources and establish an environment for restarting the failed system. While the self-healing was under way, the system also reallocated system resources to keep the order-entry workload on goal, despite effectively losing one-third of its capacity.
- *Recovering from a hardware failure*
A critical hardware component, a central processor (CP), was forced to fail, and a spare physical unit immediately took over for the failed CP without any loss of availability.
- *Nondisruptive WebSphere Application Server for z/OS upgrade*
The service level of the WebSphere Application Server for z/OS running in one of the systems was upgraded without any loss of availability.
- *Nondisruptive application upgrade*
The service level of the order-entry WebSphere application running in one of the z/OS systems was upgraded without any loss of availability.

To experience the full impact of the Mettle Test, you have to see it. To download the Mettle Test and get more information about the autonomic capabilities of WebSphere Application Server for z/OS, visit ibm.com/software/webservers/appserv/zos_os390/mettle.html.

Business value of the Mettle Test

The Mettle Test shows that WebSphere Application Server for z/OS can:

- *Manage resource consumption based on business objectives.*
- *Run diverse mixed workloads concurrently.*
- *Protect critical work from killer applications.*
- *Dynamically alter existing work priorities as business needs demand.*
- *Help enable you to use all existing capacity while potentially reducing cost.*
- *Respond quickly to shifts in business priorities.*
- *Minimize both planned and unplanned outages.*

Java performance

A high percentage of the WebSphere Application Server for z/OS runtime code is written in the Java language. The J2EE servlets and EJB are also in Java. Thus, the performance of the JDK components – JVM, just-in-time (JIT) and class libraries – is an important building block. The following section describes situations in which Java performance runs independent of a WebSphere Application Server for z/OS on J2EE container.

Integration with Java on z/OS running on System z

It might be a surprise to some people that Java on z/OS is very tightly integrated with the hardware. Figure 10 shows how Java relates to the system-control program, and how it relates to the hardware. Java applications basically consist of class files and class libraries. A Java application can certainly exploit the class library included with the JVM because this library provides all the basic constructs, like String and Hashtable, that make Java so powerful. A Java application can also take advantage of class libraries provided by various software vendors.

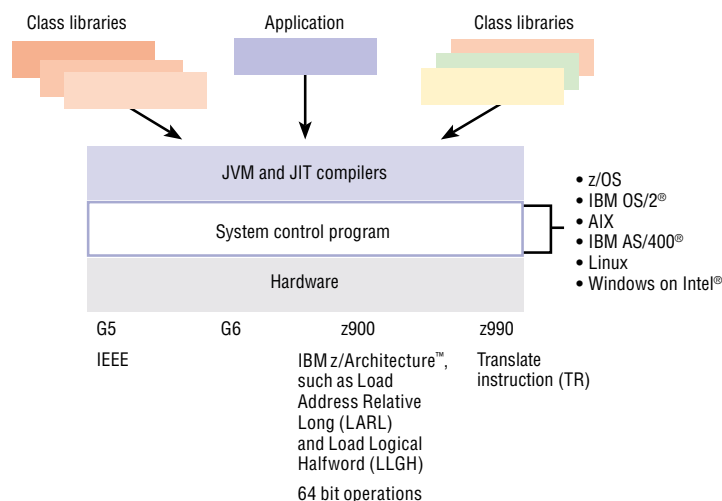


Figure 10. Java structure

Some of the support provided by the JVM must interact with the system-control program (z/OS, Linux, AIX and Windows). For example, when Java starts a new thread, underlying system-control program services are used. These portions of JVM code might be unique for a specific platform. However, most of the code in the IBM JVM is common across all the platforms for which IBM provides Java support. In addition, the class libraries provided with the IBM JVM are common across the various platforms. Therefore, most of the updates that have been made to improve the JVM for z/OS also help the JVMs for other platforms. Likewise, most of the improvements made to any other IBM JVMs benefit the JVM on z/OS as well.

All Java class files, whether they are part of the application or part of a class library, represent their implementable code using platform-agnostic Java bytecodes. Initially, bytecodes are interpreted by the JVM. This means that the JVM reads each bytecode and performs the appropriate action for that bytecode. However, after the JVM determines that Java code is used frequently enough, the JVM uses the JIT compiler to compile the bytecodes into implementable code using the appropriate instruction set for the platform. On the z/OS platform, the JIT compiler generates code using ESA/390 architecture instructions – the same code you might generate if you wrote a program in S/390 assembler language and compiled it on z/OS.

The Java JIT compiler provides some significant advantages. First, when it compiles Java code, it provides a huge increase in performance. Three-hundred and ninety instructions that are directly implemented by the hardware perform much better than JVM code to interpret each Java bytecode. Another benefit of the JIT compiler is that it knows the target machine when it compiles the code. The JIT compiler can take advantage of all the latest hardware features and instructions of the machine it's running on. The same thing is true if you run the same Java application on an old machine and a new machine – the JIT compiler running on the new machine can generate code to take advantage of the latest hardware features and instructions of that machine. This represents a significant improvement over statically compiled programs that are often compiled to run on the lowest supported hardware level.

Java performance on z/OS has improved from release to release since Java was first introduced on the platform. For historical information, as well as more current information about how Java performance on z/OS continues to improve, visit ibm.com/servers/eserver/zseries/software/java/javafaq.html#perform.

SDK performance improvements

Figure 11 shows the performance improvement from Java Software Development Kit (SDK), Version 1.3.1 to SDK, Version 1.4.2. SDK, Version 1.3.1 is recommended for use with WebSphere Application Server for z/OS, Version 5.0.2. SDK, Version 1.4.2 is recommended for use with WebSphere Application Server for z/OS, Version 5.1 and WebSphere Application Server for z/OS, Version 6.

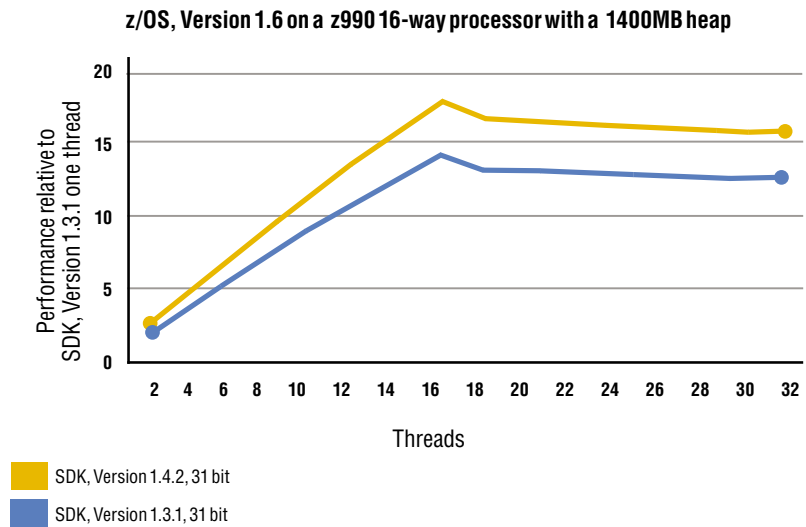


Figure 11. SDK multithreaded performance improvements

The workload used to generate the chart in Figure 11 is multithreaded and runs multiple times, each time adding another thread. The workload processes online transaction processing (OLTP) transactions like those used in the TPC-C benchmark. However, this Java workload can't provide database access or input and output. All the database information is kept in memory, in the Java heap.

Because the measurements were performed on a 16-processor machine, it's not surprising that the transaction rate increases as the number of threads is increased from 1 to 16. Each additional thread enables the benchmark to more fully use a 16-thread processor. When the seventeenth and eighteenth threads are added to the workload, other thread-management overhead causes the throughput to degrade. As more threads are added, the throughput continues to degrade, but much more gradually. Generally, these results show that Java, Version 1.3.1 and Java, Version 1.4.2 both scale very well on the z/OS operating system.

You can see that SDK, Version 1.4.2 consistently performs better than SDK, Version 1.3.1. In fact, comparing the peak, 16-thread results in Figure 2, SDK, Version 1.4.2 performs 26 percent better than SDK, Version 1.3.1. This multithreaded primitive test simulates the processing done by the JVM in a servant region and is a good indicator of improvements in WebSphere Application Server for z/OS runtime performance.

Of course, there is no guarantee that your WebSphere Application Server for z/OS application will realize the same improvement, but clearly using SDK, Version 1.4.2 rather than SDK, Version 1.3.1 can provide WebSphere Application Server for z/OS, Version 5.1 with a significant performance advantage over WebSphere Application Server for z/OS, Version 5.0.2.

WebSphere Application Server for z/OS, Version 5.1 performance

WebSphere Application Server for z/OS, Version 5.1 provided significant performance improvements over WebSphere Application Server for z/OS, Version 5.0.2. A number of factors contributed to this improvement. One key influence was how the WebSphere Application Server for z/OS run time used SDK, Version 1.4. As demonstrated in Figure 11, SDK performance improved by approximately 25 percent. The WebSphere Application Server for z/OS run time also benefited from improvements to the EJB container and higher levels of compiler optimizations.

Figure 12 illustrates a range of performance improvements from a sample of primitive benchmarks and end-to-end benchmarks. The most significant benefits are seen in MDB performance and in benchmarks, such as Trade3, that use EJBs.

WSBench tests Web services performance and is described in the section entitled “WebSphere Application Server for z/OS, Version 6.0.1 Web services performance” on page 34. The IMS eRWW benchmark consists of WebSphere Application Server accessing IMS and DB2 using the IBM IMS Connect for J2C. The CICS eRWW benchmark consists of WebSphere Application Server for z/OS accessing CICS and DB2 using the CICS Transaction Gateway J2C. The eRWW benchmark is described in the section entitled “The eRWW workload” on page 40. Approximately 40 percent of the eRWW workload is running WebSphere Application Server for z/OS and Java and the remainder is traditional CICS and DB2 or IMS and DB2 with z/OS. Thus, the 12 percent overall benefit to eRWW from WebSphere Application Server for z/OS, Version 5.1 over Version 5.0.2 is based on a sizeable improvement in the WebSphere Application Server for z/OS and Java portions of the workload.

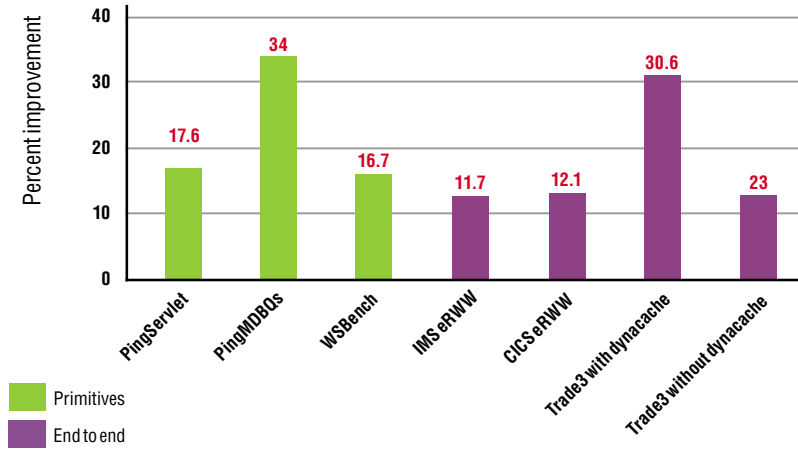



Figure 12. WebSphere Application Server, Version 5.1 performance comparisons

WebSphere Application Server for z/OS, Version 6.0.1 performance

WebSphere Application Server for z/OS, Version 6.0.1 contains a significant set of both functional and performance improvements over previous WebSphere Application Server for z/OS releases. This section of the white paper focuses on providing information that quantifies the performance improvements possible in Version 6.0.1.

Performance improvements in WebSphere Application Server for z/OS, Version 6.0.1 result from changes to the following key functional areas:

- *Improved Web container performance and scalability related to the reduction of code-path length and enhanced caching features*
- *Improved EJB container performance related to the reduction of code-path length, new read-only bean optimizations and improved read-ahead functionality*
- *A new dynamic cache called distributed map (DMap) caching that performs better than command caching because of improved cache-hit ratios*
- *Enhanced Web services performance resulting from the caching of frequently used serializers and deserializers, helping to significantly decrease the processing time for large messages*



Considerations when comparing the performance of Java and COBOL applications

- *Programmer skill*
- *Language impact*
- *Transaction-manager run times*
- *Interface impact*
- *Programming model*
- *Application design*

Although the results presented in this section highlight the improvements made to key areas of WebSphere Application Server for z/OS, they do not map directly to every application because of variations in application functionality. Performance improvements have been made across the programming model, and some of the benchmarks used in these tests exercise portions of the programming model that others do not. For example, the Web container was redesigned for Version 6, parts of the EJB container were improved, and the connection manager and JSP compiler were enhanced. As a result, any benchmark that takes advantage of these components can be expected to show performance improvements.

Trade6: A WebSphere Application Server for z/OS performance benchmark

Trade6 is the fourth generation of the WebSphere Application Server for z/OS end-to-end performance application benchmark. Trade6 models an online stock-brokerage application and has been redesigned and developed to cover the significantly expanding WebSphere Application Server for z/OS programming model. Trade6 provides a real-world application driving the WebSphere Application Server for z/OS implementation of J2EE, Version 1.4 and Web services, including key WebSphere performance components and features.

The Trade6 design spans J2EE, Version 1.4 including the EJB, Version 2.1 component architecture, MDB, transactions (one-phase and two-phase commit) and Web services. Trade6 also highlights key WebSphere Application Server for z/OS performance components such as dynamic caching, Web services, and the new Java messaging engine.

To learn more about the Trade6 workload, including download and installation instructions, visit ibm.com/software/webservers/appserv/was/performance.html.

WebSphere Application Server for z/OS, Version 6.0.1 Trade performance without dynamic caching

WebSphere Application Server for z/OS consolidates several caching activities, including servlets, Web services and WebSphere commands, into a single service called the *dynamic cache*. Caching the output of servlets, commands and JSP components can improve application performance significantly. Trade provides an option that allows the benchmark to run both with and without dynamic caching.

Figure 13 illustrates the performance of the Trade benchmark without dynamic caching. The benchmark was run on a z990 server using two processors. This measurement compares WebSphere Application Server for z/OS, Version 5.1 to Version 6.0.1, with an upgrade in DB2 Universal Database versions from Version 7 to Version 8. The DB2 KEEP_DYNAMIC¹⁰ parameter is set to YES, as opposed to past Trade measurements. The Trade benchmark is also updated from Trade3 to Trade6. Trade6 is essentially unchanged over Trade3. Trade6 uses the same database and the same business logic, with a few minor changes to accommodate some features available in WebSphere Application Server for z/OS, Version 6.0.1.

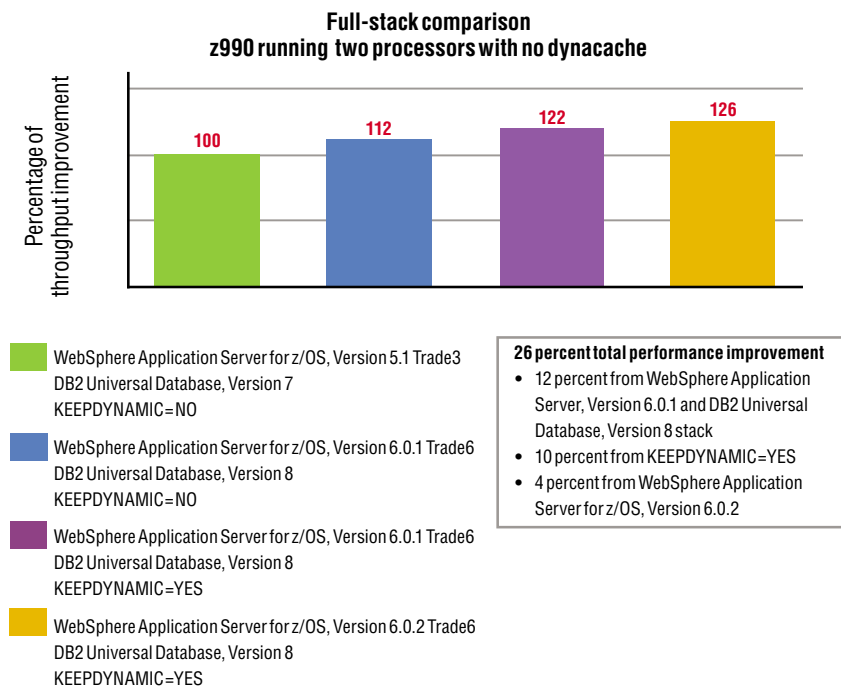


Figure 13. WebSphere Application Server for z/OS, Version 6.0.1 and 6.0.2 Trade performance without dynacache

¹⁰ The DB2 bind option (KEEP_DYNAMIC=YES) preserves dynamic statements past a commit point for an application process. An application can issue a PREPARE command for a statement once and omit subsequent PREPARE commands for that statement. The PREPARE process creates the implementable form of a dynamic SQL statement, called the *prepared statement*, from the character string form, which is called the *statement string*. When the dynamic statement cache is not active, and an application is run that is bound (with KEEP_DYNAMIC=YES), DB2 saves only the statement string for a prepared statement after a commit operation. On a subsequent OPEN, EXECUTE or DESCRIBE command, DB2 must prepare the statement again before performing the requested operation. When the dynamic statement cache is active, and an application is run that is bound (with KEEP_DYNAMIC=YES), DB2 retains a copy of both the prepared statement and the statement string. The prepared statement is cached locally for the application process. If the application issues an OPEN, EXECUTE or DESCRIBE command after a commit operation, the application process uses its local copy of the prepared statement to avoid a prepare operation and a search of the cache.

The percentage throughput improvement seen moving from WebSphere Application Server for z/OS, Version 5.1 to Version 6.0.1, and upgrading DB2 from Version 7 to Version 8 (with KEEP_DYNAMIC=YES) is approximately 26 percent. Approximately 12 percent of this improvement is the result of path-length improvements made to the Web and EJB containers. An additional 10 percent is due to the usage of KEEP_DYNAMIC. Another 4 percent improvement was delivered in WebSphere Application Server for z/OS, Version 6.0.2.

WebSphere Application Server for z/OS, Version 6.0.1 Trade performance with dynamic caching
WebSphere Application Server for z/OS, Version 6.0.1 provides an improved dynamic-cache service, providing Struts and Tiles caching and Web services client caching in addition to the servlet and JSP fragment caching, distributed map, command caching and Web services server-side caching available in Version 5.1. The WebSphere Application Server for z/OS dynamic cache is an in-memory cache that can also overflow cached entries to disk to support a large number of objects, using a hash-table-on-disk. A dynamic cache hit is generally served in a fraction of the time of a noncached request. Significant performance gains are generally achieved with Trade6 by taking advantage of the dynamic-cache technology available in Version 6.0.1. Trade6 is designed as a performance application benchmark for the WebSphere Application Server for z/OS, Version 6.0.1 dynamic cache. Trade6 integrates DMap caching as well as both command bean and a combined fragment and command-bean caching configuration for performance research and as a sample for configuring a particular application.

Figure 14 illustrates the performance differences between WebSphere Application Server for z/OS, Version 5.1 and Version 6.0.1 when using dynamic caching. When the Trade6 workload is run using command-bean caching, performance improves by 26 percent. When the same test is performed using DMap caching, performance improves by an additional nine percent. The key performance enhancement comes from improved dynacache hit performance. Rather than making copies of the object that's being referred to, you can just pass a pointer to the object and get better performance. Combining the DMap-caching enhancements with other enhancements of WebSphere Application Server for z/OS, Version 6.0.1 improves performance by 36 percent. In this case, KEEP_DYNAMIC=YES accounts for about nine percent improvement. An additional five percent improvement is delivered with WebSphere Application Server, Version 6.0.2.

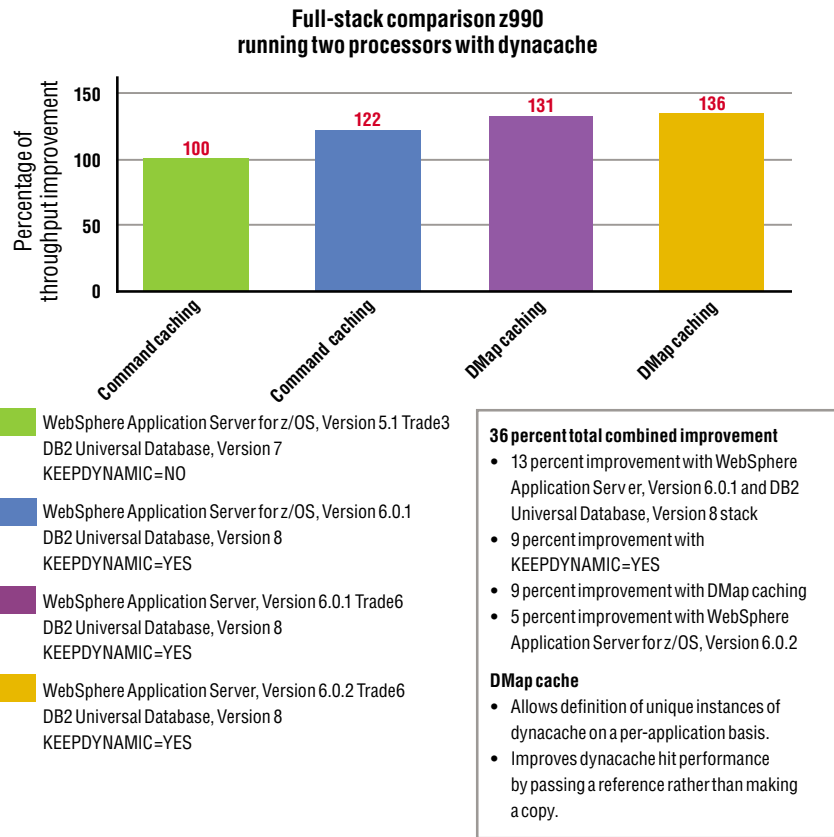


Figure 14. WebSphere Application Server for z/OS, Version 6.0.1 and Version 6.0.2 Trade performance with dynacache

WebSphere Application Server for z/OS, Version 6.0.1 Trade6 scalability (with and without dynamic caching)

The z/OS operating system and System z hardware have a long-standing reputation as being the premier platform supporting large OLTP systems. WebSphere Application Server for z/OS continues that tradition by providing excellent scalability.

In Figure 15, using the Trade6 benchmark, taking measurements without dynacache enabled and scaling the z990 server from a one-way to an eight-way engine, yields about 7.7 out of 8 engines, which indicates WebSphere Application Server for z/OS, Version 6.0.1 delivers extremely good scalability. N-way scalability tends to be a consistent pattern for the WebSphere Application Server for z/OS run time, starting with WebSphere Application Server for z/OS, Version 4 and continuing through Version 6.

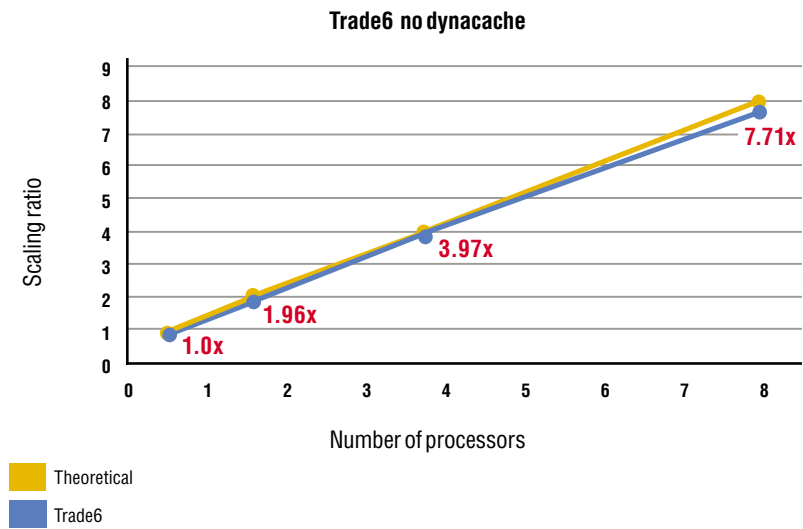


Figure 15. WebSphere Application Server for z/OS, Version 6.0.1 Trade6 scalability without dynacache

The same set of scaling measurements as shown in Figure 15 were obtained, this time using DMap caching. The tests shown in Figure 16 result in a scalability ratio of approximately seven out of eight engines, which, although representing a slight deterioration over the no-dynacache case, still show an excellent scalability ratio.

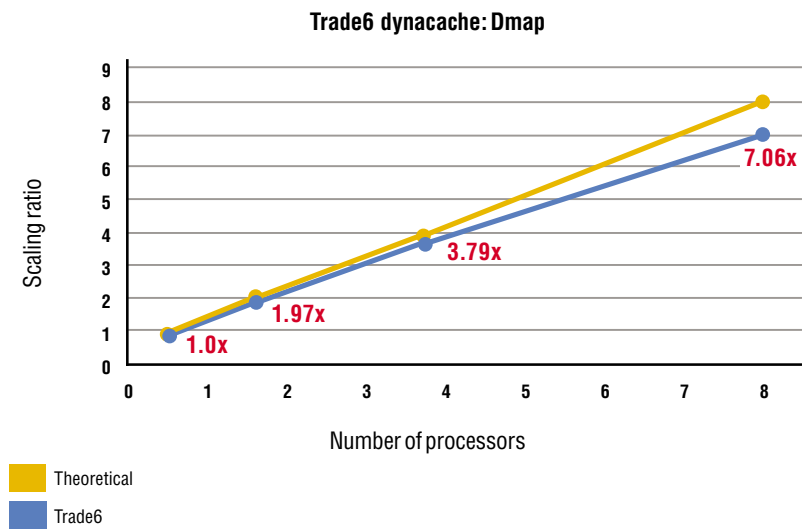
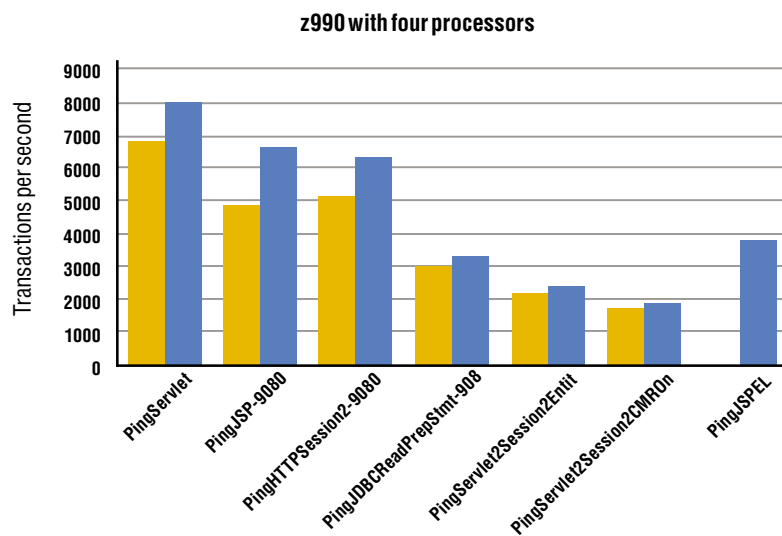


Figure 16. WebSphere Application Server for z/OS, Version 6.0.1 Trade6 scalability with dynacache

WebSphere Application Server for z/OS, Version 6.0.1 J2EE primitive performance

Component-level (or primitive) benchmarks can be used to help establish performance expectations of basic building blocks and to uncover performance bottlenecks. Trade contains more than 20 primitives that exercise commonly used J2EE operations. The measurements shown in Figure 17 were made using a number of different J2EE runtime primitives that exercise key functions of the J2EE programming model. The results show that WebSphere Application Server for z/OS, Version 6.0.1 has improved performance by from 9 to 36 percent compared to Version 5.1.



WebSphere Application Server for z/OS, Version 6.0.1 improvements from 9 to 36 percent

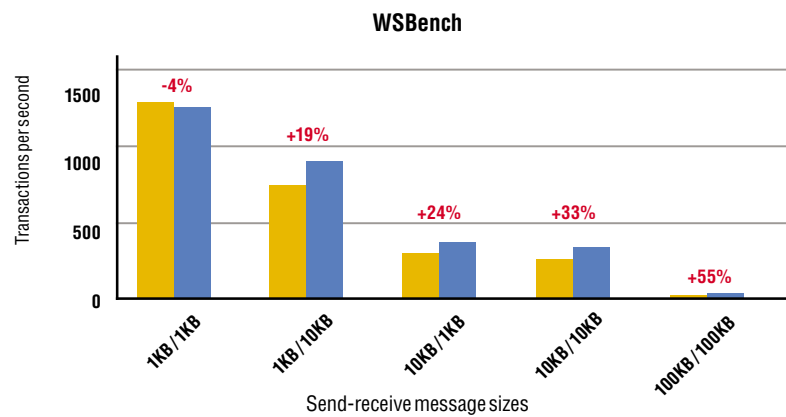
- WebSphere Application Server for z/OS, Version 5.1
- WebSphere Application Server for z/OS, Version 6.0.1

Figure 17. WebSphere Application Server for z/OS, Version 6.0.1 primitive performance comparisons

WebSphere Application Server for z/OS, Version 6.0.1 Web services performance

The WSBench benchmark provides a suite of Web services primitives with varying payload sizes, complexities and data types. The primitive suite is based on an industry-standard Web service being defined in the banking industry as well as the IBM Trade benchmark. It is built on industry standards, including SOAP, Web Services Description Language (WSDL), Java API for XML-based remote procedure call (JAX-RPC) and Web services for J2EE.

In the past, IBM has made significant performance improvements in Web services, particularly from WebSphere Application Server for z/OS, Version 4 to Version 5. Figure 18 illustrates the performance improvements in WebSphere Application Server for z/OS, Version 6.0.1 as compared to Version 5.1 using the WSBench benchmark. The scenarios vary, with the SOAP payload request and response sizes from 1 to 100 KB. Figure 18 shows significant performance improvements in Version 6.0.1, where the benefit is greater at the larger payload sizes. This is because there is a small setup penalty associated with the new enhancements, which is a larger percent of the processing for the smaller payloads. Although Figure 18 indicates a performance improvement of as high as 55 percent, the typical performance improvement expected with WebSphere Application Server for z/OS, Version 6.0.1 is approximately 30 percent.



- Significant improvements with WebSphere Application Server for z/OS, Version 6.0.1
- Percentage improvements increase with message size.

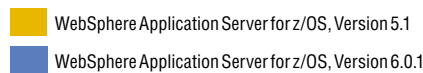


Figure 18. WebSphere Application Server for z/OS, Version 6.0.1 Web services performance comparisons

WebSphere Application Server for z/OS, Version 6.0.1 performance summary

WebSphere Application Server for z/OS, Version 6.0.1 contains a significant set of performance improvements along with functional and architectural changes that also contribute to performance benefits. Performance benefits come from a variety of sources:

- *Improved code-path and caching function in both Web and EJB containers*
- *Improved dynamic caching mechanism with DMap caching*
- *Performance improvements provided by Web services through more-efficient deserializers and caching*
- *Continued excellence in scalability*

Performance cost of Web-enabling your 3270 application

IBM clients frequently ask, “How much does Java cost compared to COBOL?” The question itself is not very specific, and neither is the answer. Comparing simple programs written in Java to a procedural language is not a fruitful exercise and is better left to an academic research paper. Usually, what is really being asked is, “How much more processing power is required to enable existing CICS or IMS transactions for WebSphere Application Server for z/OS?” or “How will the performance of a complete WebSphere and J2EE application compare to a traditional CICS or IMS application performing the same business function?”

Many factors must be considered when answering these questions. The primary intent of this discussion is to provide an overview of these issues and not to evaluate each of them in detail. Considerations include:

- *Programmer skill*
- *Language impact*
- *Transaction-manager run times*
- *Interface impact*
- *Programming model*
- *Application design*

When implementing WebSphere Application Server for z/OS either as a front end to existing transactions or to drive new application business logic using the full capabilities of the J2EE model, the entire runtime environment must be considered. The IMS, and CICS COBOL and PL/1 environments have been around for many years, and best practices are well established. Although a growing body of knowledge about J2EE best practices is available, there is still a lot of room for maturation.

Programmer skill

Based on working with clients who have implemented WebSphere Application Server for z/OS for the first time, IBM has learned that the skill and knowledge of the application developer is still a key factor. Also, when using tools such as IBM Rational® Application Developer, formerly known as IBM WebSphere Studio Application Developer, many options can have significant performance implications. Knowledge of these impacts is important. This is discussed in more detail in the section entitled “Application design” on page 39.

Language impact

In general, object-oriented languages are less easily optimized than procedural languages—although good-performing object-oriented applications are implemented. Whereas Java bytecodes can be run on any hardware architecture, JIT compilers have been highly optimized to take full advantage of underlying instruction-set architectures, including System z. In most cases, language selection is based on factors other than performance, and factors other than language selection have much more influence on performance.

Transaction-manager run times

WebSphere Application Server for z/OS, like IMS and CICS servers, is a transaction manager. IMS and CICS servers were developed during a time when processing cycles were very expensive, and thus, were highly optimized for the System z architecture. They continue to provide high performance, along with the high levels of RAS required for critical applications. IMS and CICS servers also continue to be enhanced to support a number of open, Java technology-based interfaces that enable them to participate fully in a Web services-based SOA environment.

WebSphere Application Server for z/OS has also been optimized to run on System z. However, many components of the run time have been designed to run on multiple platforms to allow cross-platform portability, and to facilitate a fully distributed programming model that requires a higher level of abstraction. Each successive release of WebSphere Application Server for z/OS provides demonstrated improvements in performance, and this trend is expected to continue. However, for the time being, WebSphere Application Server for z/OS performance has not matured to the same level as platform-specific IMS and CICS transaction managers.

Although choosing transaction managers can have performance implications, you must weigh them against the other advantages provided by WebSphere Application Server for z/OS and the J2EE application-development and deployment environment.

Interface impact

One of the key motivating factors for using WebSphere Application Server is to expose existing and new applications to the Web with an attractive GUI. When moving from an existing environment, doing this might mean a move from an IBM Systems Network Architecture (SNA) or IBM Virtual Telecommunications Access Method (IBM VTAM®) environment highly optimized for a System z infrastructure to a TCP/IP-based network environment.

GUIs usually increase the amount of data transfer and network-bandwidth requirements. For example, XML and Extensible Stylesheet Language Transformations (XSLT) provide higher levels of abstraction and increased interoperability, but they require parsing and larger data transfers. The choice of interface can have deployment considerations that can also influence performance.

On System z, local interfaces between WebSphere Application Server for z/OS, and IMS, CICS and DB2 are optimized when running within the same operating-system image. Communications between the Web container and between session beans can also be optimized when running locally to avoid a lot of serialization and deserialization overhead. However, exposing applications to HTTP, IIOP, SOAP over XML and Web services can open a business to new customers and new applications, and provide a level of business integration never before possible.

Programming model

Many J2EE programming model options can be used to implement a transaction. WebSphere Application Server for z/OS has evolved significantly from the product's early releases when only servlets and JSPs were available. Session EJBs and entity EJBs (BMPs, CMPs) provide more options and EJB specifications continue to evolve with enhancements, such as container-managed relationships (CMRs) and data-transfer objects (DTOs), among others. Connector options have also evolved. The WebSphere Application Server for z/OS run time has implemented numerous caching and locking technologies to improve EJB performance, but performance compared to functional trade-offs should still be considered.

Application design

J2EE tools can facilitate faster application development and a robust application-development and test environment. Many of the details of the underlying data structures and access methods are hidden from the developer. However, tooling technology can also affect the performance of the generated code and deployment options, which also can affect performance. Also, tooling technologies are also evolving rapidly, along with the programming model. JCA technology has moved from IBM VisualAge[®] for Java to IBM WebSphere Application Developer Integration Edition to Rational Application Developer in a very short period. This white paper discusses some of the implications to performance with these tools.

The use of HTTP sessions and EJB sessions can affect performance. Heap sizes and garbage-collection considerations must be addressed, influencing real memory-configuration requirements. How often the state is persisted, if at all, and what media is used for the persistent data, is also critical.

When using the entity-bean model, the tuning of access intents and setting database-isolation levels can have a significant influence on performance. The choice of commit scope and transaction properties can also play a performance role.

Finally, the object design of the application can facilitate good performance or seriously degrade performance. No amount of operating-system tuning, WebSphere Application Server for z/OS runtime tuning or deployment optimization can make up for a poorly designed application. These issues are not uncommon. It is recommended that the early application-design phase involve application architects who understand performance and that early proof-of-concept performance testing should be implemented prior to settling on a programming model and database design.



Details about the eRWW workload

- *Order-inventory workload*
- *A mix of seven transactions ranging from trivial to relatively heavy queries*
- *A read-write ratio of 80 to 20*
- *Run with large numbers of users (2000 or more)*
- *A seven-second user think time*

The previous discussion is far from complete, but it is clear that the question of Java versus COBOL language performance is not the critical factor. The discussion in the next section of this white paper takes a brief look at the base costs of enabling CICS transactions for WebSphere Application Server for z/OS compared to existing 3270 transactions. This white paper does not attempt to quantify all of the issues associated with Web-enabling applications. IBM continues to monitor many of these options in the lab, but the performance data is often obsolete before any analysis information can be published.

The eRWW workload

The following measurements were done using the eRWW workload that was developed in the IBM Poughkeepsie Development Lab.

The eRWW workload models an order or inventory environment. It consists of seven transactions that range from high-volume trivial transactions to relatively heavy queries. The read-write ratio is approximately 80 percent read to 20 percent write. For the 3270- and HTTP-based tests, the messages are up to 4KB in size. These messages tend to be lighter than most clients frequently experience.

The transaction behavior and mix has been adjusted to drive the System z platform in a way that is as representative as possible of the real-world environments of most clients. There is a natural skew in arrival patterns and variable selection. To be as realistic as possible, tests are run using a seven-second think time with thousands of concurrently logged-on users. All load emulation is performed externally to the system hosting the WebSphere Application Server for z/OS run time and database, and to the processor being tested.

Web-enabling your 3270 application

A 3270 CICS and DB2 environment is the base case used for the tests illustrated in Figure 19. All the business logic is in CICS with DB2 being accessed using the highly optimized CICS and DB2 interface. The initial request for a transaction, such as price quote, drives a transaction to get the CICS price-quote form, which is returned to the user. The user then fills in the fields of the form and drives the full CICS price-quote transaction by retrieving data from a DB2 system. Transaction rates count the number of CICS and DB2 transactions that have run and include the form request as part of the overall transaction. The 3270 base case was driven by a Teleprocessing Network Simulator (TPNS) across a SNA-VTAM network interface.

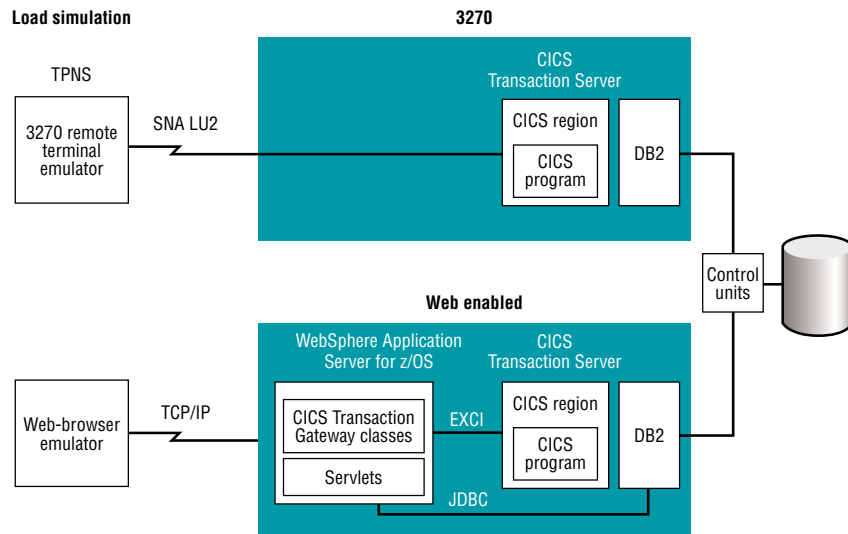


Figure 19. Web-enabling a 3270 application

The Web-enabled flow uses a Web-based emulator that drives HTTP over TCP/IP into the WebSphere Application Server for z/OS run time. As mentioned previously, the WebSphere implementation can be handled in many ways. In this case, it was compared to a servlet, session EJB and CICS Transaction Gateway model illustrated in Figure 20.¹¹

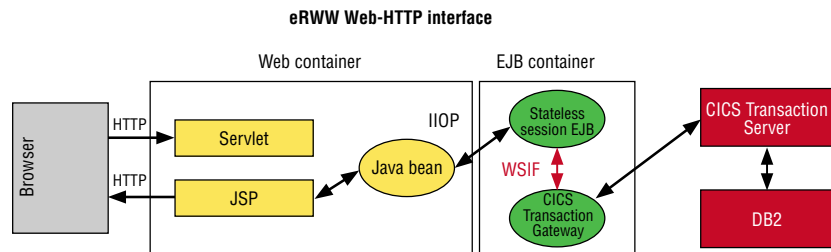


Figure 20. WebSphere Studio Application Developer Integration Edition flow for Web-enabling a CICS and DB2 transaction

¹¹ Figure 20 shows the Web Services Invocation Framework (WSIF) interface between the session EJB and the CICS Transaction Gateway connector. This is a result of using WebSphere Studio Application Developer Integration Edition tools to generate the transaction. This interface was removed with the Rational Application Developer, Version 6.0.0.1 tooling. The performance implications of this are discussed later in this white paper.

A handwritten version of a servlet model in Figure 21 was also measured, where the business logic is in the servlet instead of CICS and DB2 is accessed through JDBC. In this case, the SQL from the CICS business logic was manually (not using tools) cut and pasted into the servlet with minor modifications. In addition, this model does not allow EJB transaction properties to be defined. In the CICS Transaction Gateway example, the user can choose to allow CICS Transaction Gateway to manage the transaction-commit scope or to define the commit scope in the session EJB using a transaction property such as TRANSACTION_REQUIRED. Thus, this eRWW servlet or JDBC implementation is not fully tool-generated and is not running within a transaction-commit scope. The CICS Transaction Gateway measurements discussed in the next section were implemented using the TRANSACTION_NOT_REQUIRED property except where shown.

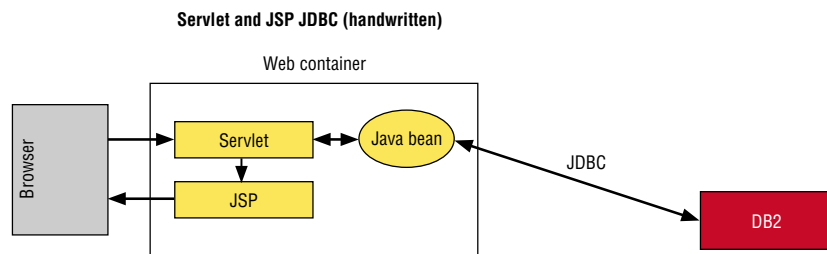


Figure 21. Flow for a manually written servlet for JDBC connection to DB2

Web-enabled eRWW performance comparisons

Many comparisons are illustrated in Figure 22. Some measurements were performed on older systems and were normalized for comparison purposes. A very close relationship exists between changes in the tools and the J2EE programming model options available to the developer, because it can affect performance. WebSphere Studio Application Developer Integration Edition tools introduced the WSIF interface, which did not exist in the VisualAge for Java tools. The WSIF interface degraded performance by as much as 25 percent in measurements performed on WebSphere Application Server, Version 5.0.2. Rational Application Developer, Version 6.0 tools eliminated this interface with the positive results discussed in the next section of this white paper. The combination of tooling improvements and WebSphere Application Server for z/OS runtime improvements have narrowed the gap with traditional 3270 performance.

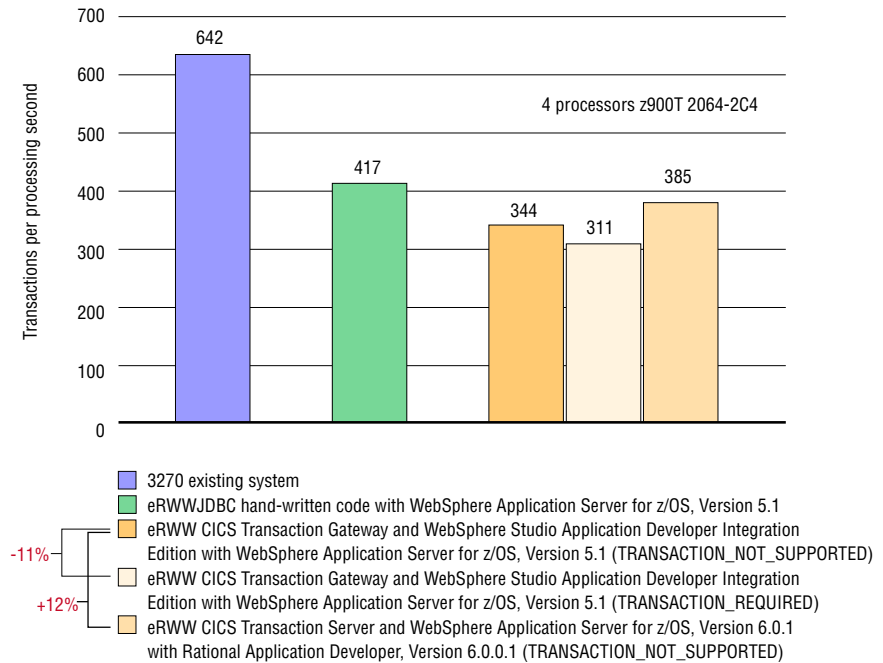


Figure 22. Web-enabled eRWW workload-performance comparisons

Figure 22 highlights a number of points about Web-enabled eRWW workload-performance comparisons:

- *Web-enabling previously existing CICS and DB2 transactions with CICS Transaction Gateway, using the latest WebSphere Application Server for z/OS and Rational Application Developer tools, can cost about 1.7 times more than a 3270 implementation.*
- *Using a customized servlet model within WebSphere Application Server for z/OS to access DB2 directly with JDBC cost 1.5 times more than a 3270 implementation. However, as previously discussed, there are tooling and transactional limitations with this model.*
- *Tooling improvements with Rational Application Developer, Version 6.0.0.1 that help eliminate the WSIF interface between the session EJB and CICS Transaction Gateway help improve performance by as much as 12 percent.¹²*
- *WebSphere Application Server for z/OS, Version 6.0.2 and WebSphere Application Server for z/OS, Version 5.1 performance using the WebSphere Studio Application Developer Integration Edition, Version 5.1 tools is equivalent (not shown here).*
- *Using TRANSACTION_REQUIRED causes an 11 percent drop in performance compared to TRANSACTION_NOT_SUPPORTED.*
- *Unfortunately, WebSphere Application Server for z/OS, Version 6.0.1 performance data for the eRWW servlet manually written implementation is not available for comparison.*

¹² Rational Application Developer, Version 6.0.0.1 improves the performance of the JCA interface by replacing WSIF with direct Common Client Interface (CCI) calls. This function helps reduce the cost of passing data over the connector. Removing WSIF also helps reduce the complexity of Java objects that are passed over the RMI/IIOP interface (not shown in the measurements discussed in this section).

The eRWW comparison is a subset of many comparisons that could be performed, and as mentioned at the beginning of this section, many factors can cause the results to vary. However, these measurements demonstrate that the baseline costs to implement WebSphere Application Server for z/OS can be less than two times the cost of previously existing non-Web-enabled 3270 CICS and DB2 COBOL and PL/I transactions.

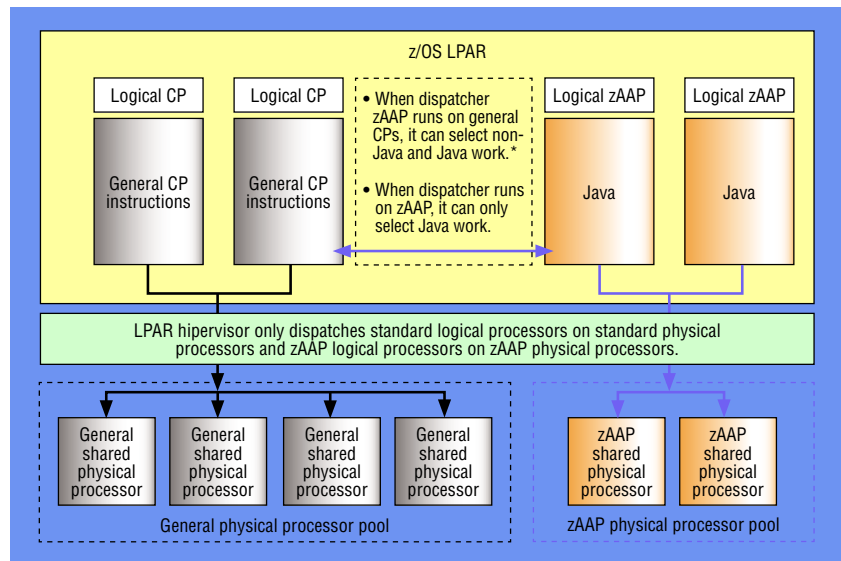
System z Application Assist Processors

As this white paper has already discussed, WebSphere Application Server for z/OS provides robust qualities of service through a unique design that takes advantage of the underlying System z hardware and operating system. Workload management, isolation, availability, security and unified systems management are built in while still providing excellent performance and scalability. System z now provides System z Application Assist Processors (zAAPs) to further enhance the WebSphere Application Server for z/OS and Java environment on z/OS to take advantage of data proximity while helping to reduce overall hardware and software costs. The following section explains what zAAPs are and how they can help improve WebSphere Application Server for z/OS price-performance ratio.

zAAP technical overview

As Figure 23 shows, a zAAP runs on a standard processing unit (PU) that can have multiple personalities. A PU can be:

- *A standard processor running z/OS, Transaction Processing Facility (TPF), IBM z/VSE™ or Linux on System z.*
- *An Integrated Facilities for Linux (IFL) system that is dedicated to run only Linux on System z.*
- *A system assist processor (SAP) that runs the input-output (I/O) subsystem code.*
- *An Interconnection Facility (ICF) that runs coupling-facility code.*



* Subject to installation controls

Figure 23. zAAP technical overview

However, with zAAPs, a PU is used to run only Java on z/OS. In this scenario, a PU can be used as a spare processor that can swap in and assume the appropriate personality if another PU with any of the previously listed personalities fails. The underlying microprocessor and instruction-set architecture is the same for all the previously listed personalities.

Modifications to the JVM for z/OS detect when Java code is called or when Java code calls out to non-Java code. These JVM modifications signal the z/OS dispatcher that the unit of work can be moved to or from a standard processor and a zAAP. These engines have also been customized to avoid some common CP functions, such as taking I/O interrupts. zAAPs share the L2 cache and main memory with the other PUs on the system. This tight integration of zAAPs with the JVM and the z/OS dispatcher, on top of logical partitioning (LPAR), performance rating (PR) and shared memory (SM) virtualization technology, helps minimize latency and processor degradation while enabling significant portions of WebSphere and Java applications to be offloaded onto engines (zAAP processors) with substantial price-performance advantages. This Java offload capability is achieved without having to modify the application. Only the systems programmer knows that the zAAPs are configured.

The memory-coherent design of System z mainframes provides a number of advantages over network-attached, distributed offload devices. It enables the operating system to manage and monitor zAAPs within the same workload manager, Systems Management Facility (SMF) and Resource Measurement Facility (RMF) infrastructure that is used to monitor and manage the other PUs. This design also delivers the full range of WebSphere Application Server for z/OS, z/OS and System z qualities of service that you expect with the System z platform – as demonstrated in the Mettle Test (see page 19).

The zAAP and the proxy for switching into the zAAP is integrated into the JVM, so that the switchover to avoid latency degradation can be efficiently managed. This support has been integrated into the z/OS dispatcher, again to help maximize performance.

zAAP requirements and characteristics

This section provides some of the requirements and characteristics for zAAPs:

- *zAAPs are available on z890, z990 and the System z9 processors with a microcode update.*
- *Java Development Kit (JDK), Version 1.4.1 or later is required to dispatch Java code to the zAAP.*
- *Java code can run on general CPs and zAAPs; however, zAAPs can process only Java.*
- *zAAP engines run at the same speed as the CPs of the z990 and System z9 machine on which they coexist. zAAPs on z890s run faster than coexisting CPs for model x10 to x60. They can run at the same speed on the x70 model.¹³*
- *Configuring zAAPs does not increase the million service units (MSUs) ratings of the CEC or LPAR. The MSU rating is determined only by the number of CPs.*
- *Any product that can run with JDK, Version 1.4.1 or later and runs on z/OS can use zAAPs. The products listed previously are just a subset of the products that can use zAAPs. WebSphere Application Server for z/OS, Version 5.1 or later must be installed to use zAAPs.*
- *The use of zAAPs is transparent to all IBM and independent software vendor (ISV) Java programs running on JDK, Version 1.4.1 or later.*

¹³ z890 models are defined as model *xyy* where *x* indicates the number of processors and *yy* indicates the processor power rating. Thus, a model 110 is a uniprocessor at the lowest power rating. A model 170 is a uniprocessor at the highest power rating. The high end of the product line is the model 470—four processors at the highest processor power rating.

zAAP price-performance advantages

Price-performance savings from zAAPs come from three main sources:

- *Hardware savings. zAAPs cost US\$125 000 dollars per PU (engine) and are priced similarly to Integrated facilities for Linux® (IFLs).*
- *Software savings. IBM and ISVs historically charge for software based on the MSU rating of the CEC or LPAR. MSU ratings are based on the number of CPs only. zAAP capacity is not included in the MSU rating. ISV pricing policies vary from vendor to vendor and are the prerogatives of each vendor. It is IBM policy to not charge for IBM software on zAAPs. At the date of publication of this white paper, IBM is not aware of any vendors who charge for software running on zAAPs.*
- *Configuration savings. Prior to zAAPs, many clients were configuring WebSphere Application Server for z/OS in an LPAR separate from the back-end data source to help minimize software costs. This is no longer necessary because adding zAAPs to enable running WebSphere Application Server for z/OS in the same tier of an existing system does not usually increase the cost of the preexisting software. At the same time, it offers the performance advantages of data proximity and helps to reduce the number of operating-system images that need to be managed.*

Table 2 illustrates some of these points with examples. These examples are not migration scenarios.

Configuration	Reduction in MSUs	MSU cost savings	CP cost savings
Traditional two-way processor compared to one general processor (GP) and one zAAP (1 + 1)	48 percent fewer software pricing MSUs (132 compared to 70 MSUs)	<ul style="list-style-type: none"> • WebSphere Application Server for z/OS costs reduced by 43 percent • Typical monthly licensing charge (MLC) stack costs reduced by 32 percent 	US\$125 000 per zAAP compared to traditional hardware costs
Traditional 16-way processor compared to eight GPs and eight zAAPs (8 + 8)	41 percent fewer software pricing MSUs (761 compared to 448 MSUs)	<ul style="list-style-type: none"> • WebSphere Application Server for z/OS costs reduced by 32 percent • Typical MLC stack costs reduced by 26 percent 	US\$125 000 per zAAP compared to traditional hardware costs
Traditional three-way processor compared to two GPs and one zAAP (2 + 1)	31 percent fewer software pricing MSUs (191 compared to 132 MSUs)	<ul style="list-style-type: none"> • WebSphere Application Server for z/OS costs reduced by 27 percent • Typical MLC stack costs reduced by 22 percent 	US\$125 000 per zAAP compared to traditional hardware costs

Table 2. Examples of theoretical z990 configuration cost savings

More savings can potentially be realized when an organization upgrades to future, faster processors. At the date of publication for this white paper, the cost of IFLs have been held constant for the newer System z9 platform.

Notes:

1. MLC stack: z/OS with features, IMS, Version 7, DB2 Universal Database, Version 7, CICS Transaction Server, Version 2 and IBM COBOL for z/OS, Version 2
2. WebSphere Application Server, Version 5: License cost, one year service and subscription (S&S)

These ratios can vary depending on the size of the system and the ratio of CPs to zAAPs, which is primarily determined by the nature of application itself.

zAAP and CP configuration options

For a given number of PUs in a configuration, a number of combinations of CPs and zAAPs are possible, as illustrated in Figure 24. Configuring more zAAPs than CPs on a CEC is not permitted. It is possible to have more zAAPs than CPs within a LPAR as long as the physical ratio of zAAPs to CPs does not exceed 50-50 on the CEC.

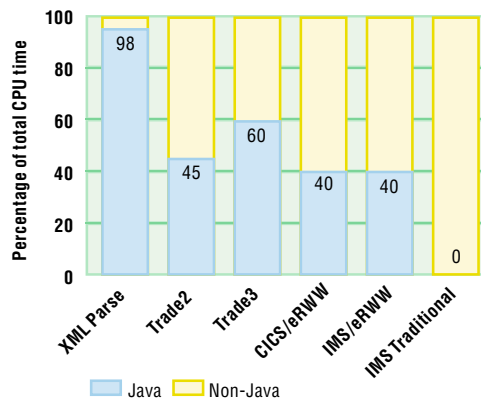
**Possible combinations CPs and zAAPs
per number of processors**

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
	1+1	2+1	3+1	4+1	5+1	6+1	7+1	8+1	9+1	10+1	11+1	12+1	13+1	14+1	15+1
			2+2	3+2	4+2	5+2	6+2	7+2	8+2	9+2	10+2	11+2	12+2	13+2	14+2
				3+3	4+3	5+3	6+3	7+3	8+3	9+3	10+3	11+3	12+3	13+3	14+3
					4+4	5+4	6+4	7+4	8+4	9+4	10+4	11+4	12+4	13+4	14+4
								5+5	6+5	7+5	8+5	9+5	10+5	11+5	12+5
										6+6	7+6	8+6	9+6	10+6	11+6
												7+7	8+7	9+7	10+7
															8+8

Figure 24. zAAP configuration options

zAAPs and performance

In the lab, IBM evaluated zAAP performance with a number of different workloads with a range of Java characteristics. Figure 25 shows a few examples of the Java content for these workloads.



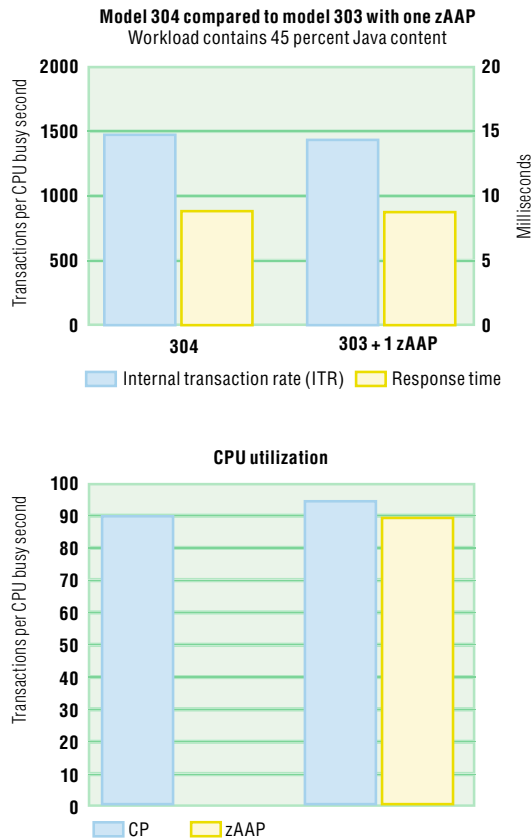
- **XML Parse:** processing-intensive XML, no WebSphere Application Server. Parses documents of different sizes with Simple API for XML (SAX) and Document Object Model (DOM), both with and without validity checking.
- **Trade2:** WebSphere Application Server, JDBC and DB2. Simulates brokerage work. Includes session servlets, JSP, session EJB, CMPs and light SQL.
- **Trade3:** WebSphere Application Server, JDBC and DB2. Demonstrates evolution of Trade2 to EJB, Version 2.0 and J2EE, Version 1.3. Includes MDBs and publish-subscribe.
- **CICS/eRWW:** WebSphere Application Server, CICS Transaction Gateway, CICS Transaction Server and DB2. Based on WebSphere Application Server technology-enabled existing OLTP applications, including HTTP, servlets, JSP, session EJB and CICS business.
- **WebSphere Application Server, IMS Connector for Java, IMS and DB2.** Based on WebSphere Application Server technology-enabled existing OLTP applications, including HTTP, servlets, JSP, session EJB and IMS business logic.

This information is for demonstration purposes only. Your workload might be different.

Figure 25. Sample workloads and percentage of Java content

The percentages in Figure 25 are based on the total end-to-end path length of the transaction. If WebSphere Application Server for z/OS is configured to run its own LPAR accessing data remotely, the percentage of Java content within the WebSphere Application Server for z/OS LPAR would be considerably higher: 75 to 90 percent is not uncommon. Pages 53 and 54 provides information about how to determine the percentage of Java content for your workload.

Figure 26 shows the impact of running with zAAPs and the overhead associated with dispatching between CPs and zAAPs.



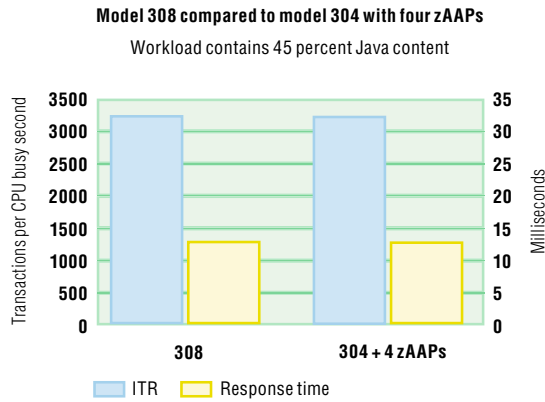
ITR dropped 1.1 percent with equivalent response time

Figure 26. Performance impact of zAAPs on z990 systems

In this example, a z990 four-CP system is compared to a z990 three-CP system with one zAAP. A WebSphere Application Server for z/OS and DB2 workload ran with approximately 45 percent Java content. The impact to throughput and response time is minimal. There is a one percent increase in the processing costs per transaction to accommodate the costs of the additional dispatching. In this case, a significant amount of Java processing is still running on a CP because the workload is 45 percent Java with only 25 percent of the capacity available on a zAAP. The z/OS dispatcher sends Java work optimally to both CPs and zAAPs depending on the dynamic characteristics of the workload and the configuration.¹⁴

¹⁴ Two SYS1.PARMLIB members can be set to specify how Java work can be dispatched between zAAPs and CPs, and whether the workload-management priority of the Java work can be honored when running on CPs. It is not in the scope of this paper to go into detail about this contingency. For a discussion of these parameters and many other zAAP capacity-planning issues, refer to the zAAP capacity planning white paper at ibm.com/support/tech-docs/atmsastr.nsf/Webindex/WP100417.

Figure 27 shows a similar comparison using the same workload comparing a z990 eight-CP system to a z990 four-CP system with four zAAPs.



ITR dropped 2.8 percent with equivalent response time

Figure 27. Performance impact of zAAPs on z990 systems

In this case, the processing cost per transaction to manage the dispatch activity between CPs and zAAPs is close to three percent. Numbers that constitute a slight performance improvement with zAAPs can be seen as high as five percent (sometimes cache benefits can be realized from dispatching the Java work on a reduced set of processors). Most measurements are in the two-to-three percent range. Response time is not noticeably affected when the system is configured properly. These increases in processing cost are minimal when weighed with the overall price-performance benefits provided by zAAPs, and should not be viewed as a deterrent to taking advantage of zAAPs.

zAAPs and z890 models

zAAPs on z890 models can run at a different speed than the CPs on z890, depending on the model of z890 used. This difference in zAAP speed compared to that of CPs could be attractive for clients with Java workloads with high Java content. Seven gradients of processor power are available on z890s from the model x10 to the model x70.¹⁴ The power ratio for a model 170 compared to a model 110 is approximately 13.8 times.

When adding zAAPs to CPs on a z890, the zAAP always runs at the full-rated speed of a model 170 engine (minus symmetric multiprocessor [SMP] effects). Thus, it is possible for a zAAP on a model 410 to be 13.8 times more powerful than the combination of the three CPs on the System z box.

Figure 28 illustrates the performance behavior as a result of the mixed-processor speed factor. In this example, a model 410 with four CPs is compared to a model 310 with three CPs and one zAAP. This configuration provides a large net increase in the overall capacity of the four-way z890. The performance comparison shown in Figure 28 indicates the throughput increases by almost two times and response time is reduced by almost half when substituting a zAAP for a CP. In this case, the zAAP engine was only running at 20 percent busy. The delta between all CPs and a CP and zAAP configuration can diminish as the power rating of the z890 increases. On a model 470, there would not be a significant change in performance.

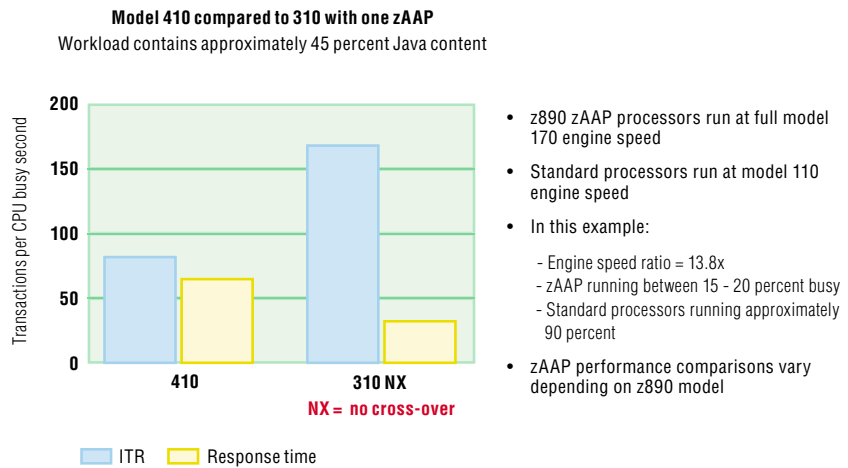


Figure 28. zAAPs and z890

Determining how many zAAPs you need

To determine the number of zAAPs you need, you must consider the following aspects:

- *The amount of zAAP-eligible content in a workload*
- *That zAAPs can write an output message every five minutes (but that figure is adjustable)*
- *That zAAPs are integrated as part of SDK, Version 1.3.1 SR24 or later*
- *That zAAPs are integrated as part of SDK, Version 1.4*

The Java SDK product provides the capability to measure and report the Java content of a workload, independent of any special z/OS level. This is very useful in estimating the extent of Java processing that can use a zAAP. Also, if you are already running on z/OS, Version 1.6 (but have not ordered any zAAPs yet), the zAAP-eligible content of a workload can be measured and reported in the RMF workload activity report. Similar to the Java SDK capability, this data in the RMF report can be used to estimate the potential zAAP load of the system.

How to monitor zAAPs in an operational environment

The measurement of zAAP processing has been fully integrated into standard RMF monitoring and reporting. Just like standard CPs, zAAP busy utilizations are reported in the processing-activity report. The workload-activity report also provides more-granular information on zAAP usage for workload, service-class and report-class entities by extending the postprocessor CPU activity report, the postprocessor workload report and the Monitor III enclave report. RMF provides the ability to:

- *Distinguish between standard CP and Integrated File Adapter (IFA) processors where necessary.*
- *Collect and report about IFA service times.*
- *Collect and report about IFA using delay states for service- and report-class periods.*

The following SMF record types are extended:

- *SMF record 70 subtype 1 (processing activity)*
- *SMF record 72 subtype 3 (workload activity)*
- *SMF record 79 subtypes 1 and 2 (address-space state and resource data)*

For more information about zAAPs

A variety of information and resources are available about zAAPs, including:

- *The zAAP projection tool for Java 2 Technology Edition, SDK, Version 1.3.1, available with the Microsoft Excel Summary Workbook. This tool runs in a test environment and gathers usage information about the percentage of Java content in your workloads that could run on a zAAP. It is useful in predicting the number of zAAPs necessary for optimum configuration. You can find more information about this tool at ibm.com/support/techdocs/atmastr.nsf/WebIndex/WP100417.*
- *The z/OS Performance: Capacity Planning Considerations for zAAP white paper, which describes the zAAP projection tool, prototype measurements and the capacity planning methodology. To download this white paper, visit ibm.com/support/techdocs/atmastr.nsf/WebIndex/WP100417.*
- *The IBM zAAP Web site is available at ibm.com/servers/eserver/zseries/zap/gettingstarted/.*

Summary

The key points of this white paper include:

- *WebSphere Application Server for z/OS, Version 5.1 performance improvements. A steady stream of performance improvements have occurred since the first WebSphere Application Server for z/OS, Version 4.0 release. WebSphere Application Server for z/OS, Version 5.1 has demonstrated performance improvements across the board in many different workload environments. Many of these improvements are influenced by the improved performance of JDK, Version 1.4.2 over JDK, Version 1.3.1.*
- *WebSphere Application Server for z/OS, Version 6.0.1 performance improvements. WebSphere Application Server for z/OS, Version 6.0.1 shows improvements in primitive test cases from 9 to 36 percent. New tooling, programming model enhancements and tuning options combine to improve end-to-end performance. And improvements have been made in Web services, particularly for large documents.*
- *High-quality n-way performance. From ping servlet to Trade3 to client benchmarks with Parallel Sysplex, the WebSphere Application Server for z/OS run time consistently demonstrates excellent scalability with n-way ratios that are better than the traditional LSPR workloads.*
- *WebSphere Application Server for z/OS optimization for performance when configured in close proximity to data. Type-2 JDBC optimization to DB2, as well as pass-by-reference local optimization, helps reduce cycles and improve response times.*

- Value-added WebSphere Application Server for z/OS runtime features performance improvements. *A two-phase-commit capability is integrated into the operating system with RRS. Workload-management capabilities are tightly integrated into the mainline path of the run time. And the product is integrated with Parallel Sysplex to provide availability, scalability, resource and systems-management advantages.*
- Significant price-performance value of zAAPs, without losing qualities of service. *zAAPs build on the tightly integrated WebSphere Application Server for z/OS, z/OS and System z systems structure to help reduce hardware and software costs for running WebSphere Application Server for z/OS and Java on z/OS. zAAPs also facilitate running WebSphere Application Server for z/OS in close proximity to data to maximize qualities of service.*
- Many factors to consider when evaluating performance of WebSphere Application Server for z/OS and Java compared to COBOL, PL/I, and CICS and IMS. *The baseline costs to implement WebSphere Application Server for z/OS are less than 2 times compared to existing CICS and COBOL, and PL/I 3270 transactions.*
- High availability and resource-management operational demos. *See the Mettle Test at ibm.com/software/webservers/appserv/zos_os390/mettle.html. Or read the ITSO Redbook for High Availability SG24-6850 at www.redbooks.ibm.com/redpieces/abstracts/redp3968.html.*
- A growing set of performance monitoring and profiling tools. *Learn more about these in the ITSO Redbook SG24-6825 at www.redbooks.ibm.com/redbooks.nsf/Redbooks?SearchView&Query=sg24-6850&SearchMax=4999.*

For more information

To learn more about IBM WebSphere Application Server for z/OS, contact your IBM representative or IBM Business Partner, or visit:

ibm.com/software/websphere/appserv

To learn more about zAAPs, contact your IBM representative or IBM Business Partner, or visit:

ibm.com/servers/eserver/zseries/zaap/gettingstarted/

To join the Global WebSphere Community, visit:

www.websphere.org



© Copyright IBM Corporation 2006

IBM Corporation
Software Group
Route 100
Somers, NY 10589
U.S.A.

Produced in the United States of America
05-06
All Rights Reserved

AIX, AS/400, CICS, DB2, DB2 Universal Database, @server, IBM, the IBM logo, IMS, MVS, OS/2, Parallel Sysplex, RACF, Rational, S/390, System p, System z, System z9, VisualAge, VTAM, WebSphere, z/Architecture, z/OS, zSeries, z/VM and z/VSE are trademarks of International Business Machines in the United States, other countries or both.

Intel is a trademark of Intel Corporation in the United States, other countries or both.

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

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

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

Other company, product and service names may be trademarks or service marks of others.

References in this publication to IBM products or services do not imply that IBM intends to make them available in any other countries.

The information contained in this document is provided AS IS. Any person or organization using the information is solely responsible for any and all consequences of such use. IBM accepts no liability for such consequences.

Any performance data contained herein was determined in a controlled environment. Therefore, the results obtained in other operating environments may vary significantly. Some measurements may have been made on development-level systems and there is no guarantee that these measurements will be the same on generally available systems. Furthermore, some measurements may have been estimated through extrapolation. Actual results may vary. Users of this document should verify the applicable data for their specific environment.

These prices are subject to change without notice and do not include applicable sales taxes. IBM is not responsible for printing errors which result in pricing or information inaccuracies. These prices are for informational purposes only and do not limit in any way a remarketer's ability to set its own price for IBM products.

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

This paper discusses strategy and plans, which are subject to change because of IBM business and technical judgments.

This paper provides addresses to non-IBM Web sites. IBM is not responsible for the content on such sites.