High Availability Architectures For Linux on IBM System z

March 31, 2006

Contents

Authors

Steve Wehr
IBM System z New Technology Center

Scott Loveland
IBM System z Linux Integration Test

Harriet Morril
eServer High Availability Center of Competence

Scott Epter
eServer High Availability Center of Competence

Abstract

This paper combines the efforts and talents of the IBM System z New Technology Center, the eServer High Availability Center of Competence, and Linux on System z Integration Test to produce a set of reference architectures that provide High Availability for applications running on Linux for System z.

This paper focuses on those architectures that cover the following scenarios:

- Where the application runs on Linux virtual servers under z/VM. The database may be on Linux for System z or on z/OS.
- Of highest interest to our customers.
- Unique to System z. Not covered are scenarios and architectures that have already been documented on other distributed platforms. WebSphere HA has been extensively covered in many documents. Although our architectures will use HA features of WebSphere, we will not concentrate on documenting WebSphere. (Please see the references section for more WebSphere HA documentation). Rather, we will concentrate on the HA aspects of the database servers that WebSphere applications would use and how System z HA features can benefit database servers.
- Have not been documented before on System z.

This paper does not cover:

- All the details necessary to implement the reference architectures. For those details, please refer to "*System z Platform Test Report for z/OS and Linux Virtual Servers*" written by the IBM Poughkeepsie Test and Integration Center for Linux.
- HA networking considerations. We cover the major components and flow between them. We have not covered how to create a highly available network.
- How to HA-enable your storage subsystem.

Introduction. Definition of High Availability

For the purpose of this paper, we have adopted the definition used by the HA Center of Competence in Poughkeepsie, NY.

- **High Availability –** Designed to provide service during defined periods, at acceptable or agreed upon levels, and masks unplanned outages from end-users. It employs Fault Tolerance; Automated Failure Detection, Recovery, Bypass Reconfiguration, Testing, Problem and Change Management
- **Continuous Operations (CO) --** Designed to continuously operate and mask planned outages from end-users. It employs non-disruptive hardware and software changes, non-disruptive configuration, and software coexistence.
- **Continuous Availability (CA) –** Designed to deliver non-disruptive service to the end user 7 days a week, 24 hours a day (there are no planned or unplanned outages).

Our architectures strive to provide **Continuous Availability.** Note that in some architectures this is not possible due to delays in the automated recovery of some system components. These delays can be long enough to cause user transactions to fail and have to be re-entered.

Chapter 1: Introduction to High Availability with z/VM and LPARs

When Linux runs on distributed architectures it is often running directly on the hardware of a single server. Although pSeries servers can now have logical partitions, their virtualization capabilities are not as extensive as System z, where z/VM allows all system resources to be dynamically shared.

Linux on System z is always running in a logical partition (LPAR). So we have introduced two new layers between Linux and the hardware, namely z/VM and LPAR. These layers play prominently in the availability of your applications, because they provide services that the Linux systems use.

Where are the Single Points of Failure (SPoFs)?

Consider an example where a System z server has several LPARs running z/OS, and one LPAR running z/VM to host Linux guests. You have installed an application on a single Linux server. Where are the points of failure? There are several:

- The System z hardware could experience multiple unrecoverable failures, causing the entire server to fail.
- The disk subsystem could fail. Note that this paper does not include any information on HA-enabling the disk subsystem.
- The LPAR microcode could fail.
- z/VM could fail.
- Linux could fail.
- Application A could fail.

The odds of each failure are different. In this case, the probability of an application failure is highest, while the probability of the System z hardware failure is lowest. The others fall on a continuum between those extremes.

So how do we eliminate these single points of failure? An easy and effective method is to eliminate them by duplicating them. Duplicating the application is usually easy, but duplicating the System z hardware can be expensive, with the cost and difficulty of the others falling on a continuum between these extremes.

The following table summarizes these points:

Single Point of Failure	Probability of Failure	Cost to fix SPoF
System z hardware	Very Low	High
Disk Subsystem	Very Low	Medium
LPAR	Very Low	Low
z/VM	Low	Low
Linux	Low	Very Low
Application	High	Very Low

Besides hardware and software failures, the following can also cause downtime for the application:

- System z hardware upgrades requiring Power On Reset POR
- LPAR configuration changes requiring reboot of the LPAR
- z/VM maintenance
- Linux kernel maintenance that requires reboot
- Application maintenance

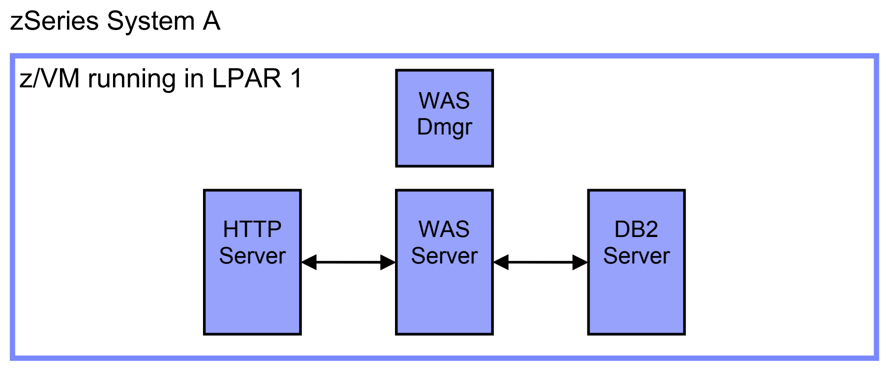
There are no probabilities that can be assigned to these since they are directly under the control of the customer. The customer's policies will dictate how often these will occur.

In order of increasing availability, the following examples examine some possible architectures and their single points of failure.

Example 1: High Availability not needed

In this example, an application is installed on a single Linux server that runs under z/VM. The SPoFs for the application are:

- System z hardware
- LPAR
- z/VM
- Linux
- Application



Each small box represents a virtual Linux server running as a guest of z/VM, in a single z/VM LPAR

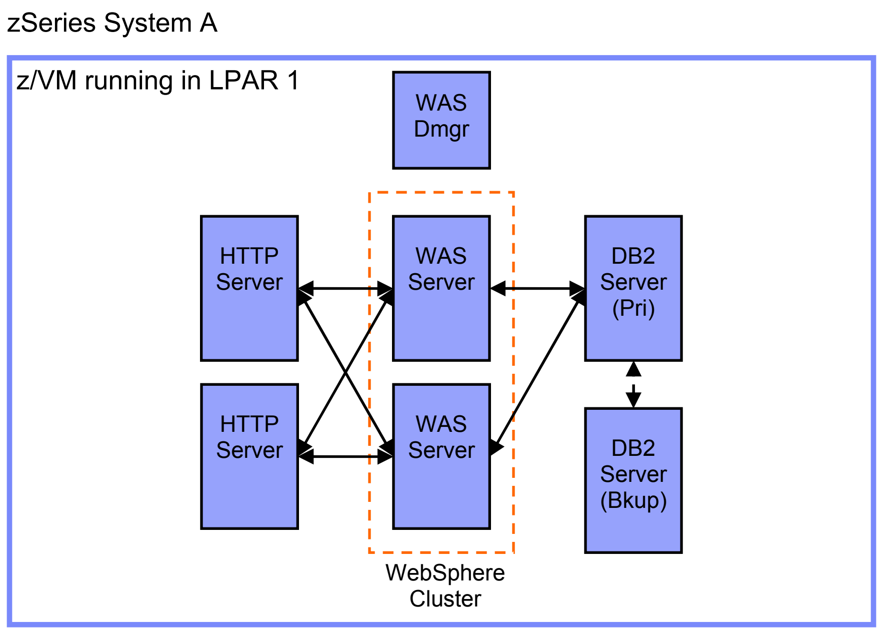
The most likely to fail is the application. When this happens, the application can be restarted and recovery time will be a few minutes. Or perhaps Linux has to be rebooted and recover time could be around 5 minutes. Recovery requires that some manual or automatic method be implemented to detect the failure and initiate recovery.

If this recovery time is sufficient then there is nothing more that needs to be done. If not then higher availability is needed.

Example 2: Moderate Availability Needed

In this example, the application is installed on multiple Linux servers that run under z/VM. The SPoFs for the application are reduced to:

- System z hardware
- LPAR
- z/VM



Each small box represents a virtual Linux server running as a guest of z/VM, in a single z/VM LPAR.

By simply replicating the application across two or more Linux servers, we have removed the most likely points of failures. Workload is distributed to the duplicated servers so that a failure in any server still leaves another server available to process the workload.

A failure in any Linux server still allows the application to run with the full resources available to it before on the remaining virtual servers. This feature is unique to Linux on System z. Because all of the CPU and memory is shared among the Linux servers under z/VM, a failure of one Linux server or application frees up its memory and CPU for use by other Linux servers. For example, if the two WAS servers are both 80% CPU utilized (using 80% of one CPU), then if one of them fails the other can increase it's CPU utilization to 80% of two CPUs. This allows the remaining server to process the full workload immediately, without having to reconfigure the server.

This is very different from failover scenarios in distributed architectures, where each server must be sized to handle significantly more than its own workload so that the server can have the capacity to handle additional workload if another server fails.

Why use only two WebSphere servers? On System z there is usually little reason to use more than two production servers in a WebSphere cluster. Usually the entire workload can be accomplished with one server; the second is added only for failover. In Linux on System z, adding more virtual servers does not add any more processing resources (CPU, memory) to the application, but instead makes z/VM work harder to run all of the production servers in memory simultaneously. For these reasons we recommend only two production servers.

Some bottlenecks do exist that can be lessened by duplicating the WebSphere application server. Application Servers can be cloned either horizontally (on another Linux server) or vertically (on the same Linux server). Bottlenecks that can be helped by this include:

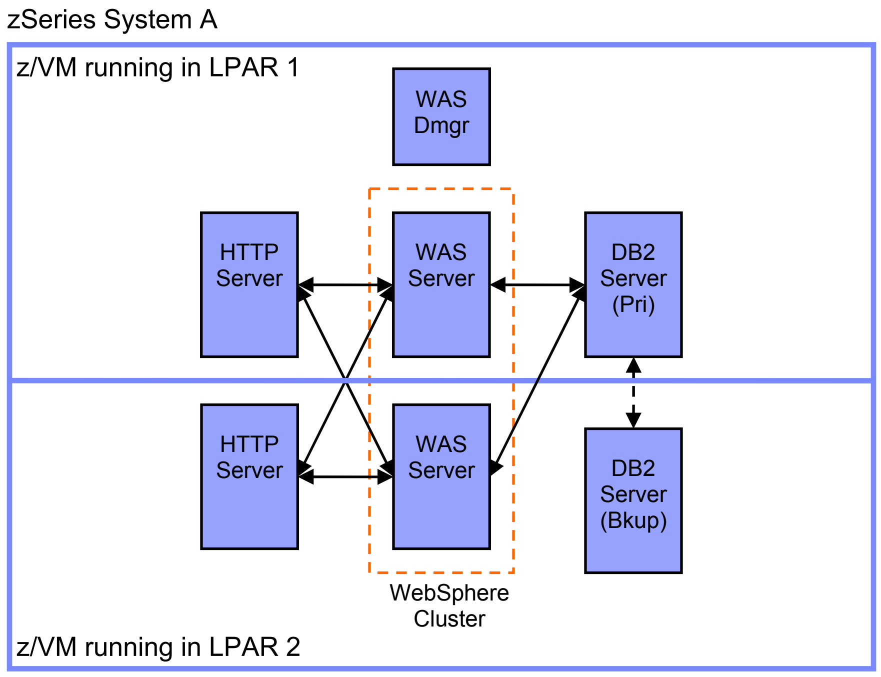
- Not enough JVM heap to run the application at the desired workload
- Not enough connections in the connection pool

Another consideration for running two WebSphere servers is that if one is unavailable due to either a planned or unplanned outage, you have only one server left to handle the entire workload. This is usually not a problem on Linux on System z because all of the resources (CPU and memory) that were available to both servers before the failure are now available to the remaining server. This does leave you with a single point of failure, however, during the time when one of the servers is unavailable. For these reasons, it is recommended that you use three servers instead of two when higher availability is required.

Example 3: High Availability Needed

In this example the application is installed on multiple Linux servers that run under multiple z/VM systems on multiple LPARs. The SPoFs for the application are reduced to:

- System z hardware



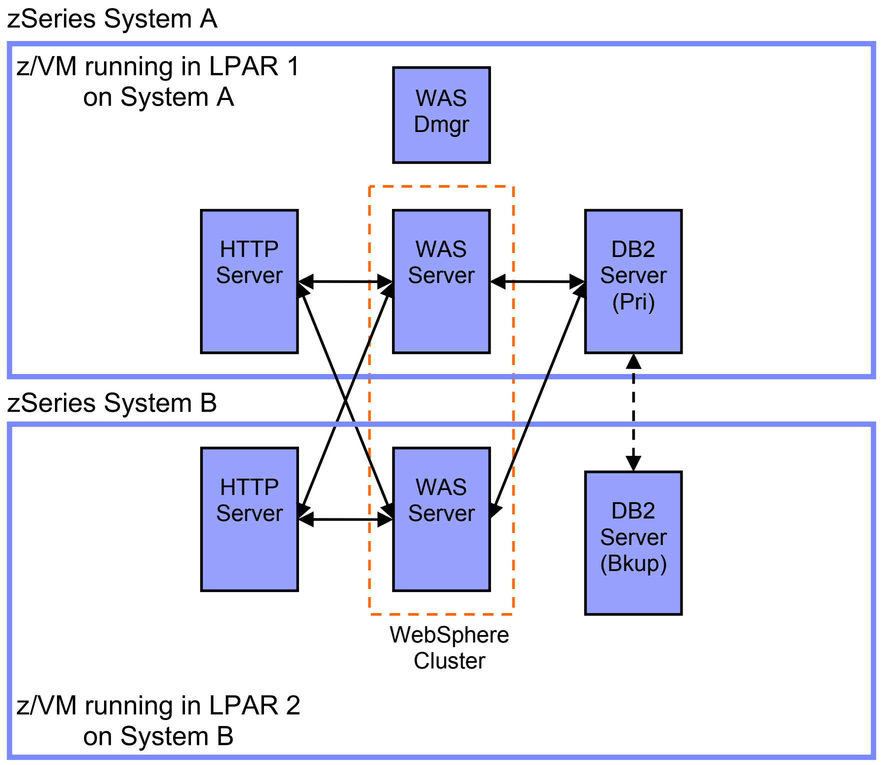
We have eliminated most of the SPoFs by creating a second LPAR that will run z/VM and Linux guests. The application is installed on Linux servers in each LPAR. The cost for doing this is still low, since both LPARs will share the same IFLs, and the real memory can be split between the two LPARs.

A failure in the application, the Linux server, VM, or LPAR still allows the application to run on the remaining virtual servers with the full resources available to it before the failure. Should one server, VM, or LPAR fail, the other LPAR can use all of the IFLs that were being shared by both.

Because you are running the same software on the same number of IFLs, software costs do not increase. For all these reasons, this is one of the most cost-effective High Availability architectures for Linux on System z.

Example 4: Continuous Availability Needed

In this example, the application is installed on multiple Linux servers that run under multiple z/VM systems on multiple LPARs on multiple System z servers. No SPoFs for the application remain.



We have eliminated the SPoFs by using a second System z server (System B) to host our second LPAR that will run z/VM and Linux guests. The application is installed on Linux servers in each LPAR. During normal operations, each LPAR receives 50% of the workload of the application.

However, the cost of adding a second System z server is to run 100% of the workload should the other server fail. Software costs increase because of this.

It is more cost-effective to run a System z LPAR near 100% CPU utilization. But the above architecture would run each LPAR nearer to 50% utilization, so that it has extra capacity in case of a failure of the other LPAR. Some alternatives to bring the utilization nearer to 100%:

- Configure fewer IFLs than are needed to run 100% of the workload in the LPAR. Configure other IFLs as standby IFLs that can be brought online quickly with Capacity Upgrade on Demand. When extra capacity is needed:
  1. The standby IFLs are defined as “active” to the LPAR.
  2. VM varies the new IFLs online.

  This process is non-disruptive and can be automated or completed manually in a few minutes.

- Run other lower-priority work in each LPAR. Configure the Linux guests so that those running the WebSphere workload have a higher priority than those running the “other” work. If a failover occurs VM will give the system’s CPU and memory to the WebSphere guests and withhold CPU and memory from the other workloads.

Summary

Examples 2 - 4, and all of the reference architectures in this paper, use clusters containing two members. You can always choose to instead create a cluster of *three* servers. In our examples that use one server type per LPAR, you could instead define three LPARs. The advantage of a three-member cluster over a two-member cluster is that should one cluster member fail, you still retain a cluster of two members, and a good degree of High Availability. With a two-member cluster, if one member fails then you are now running in non-HA mode on the single remaining member, until the failed cluster member can be brought back online.

When the absolute highest levels of availability must be maintained at all times, it is recommended to use a cluster of three LPARs. Otherwise a cluster of two LPARs is sufficient.

The above exampleSection “Example 3: High Availability Needed”,shows the most cost-effective solution for architecting LPARs and VM for High Availability. The following is recommended:
- Use a single System z server so that your z/VM and Linux LPARs can share the same IFLs.
- Use two LPARs to run your production workload.
- Create clusters of applications split between Linux servers running in each LPAR.
- Run your test and development Linux servers either in:
  - Their own LPAR. You can use the following LPAR weights as starting values:
    - Production1: 35%
    - Production2: 35%
    - Test/Dev: 30%
  - One of the two production LPARs. Give that LPAR more resources than the other production LPAR. You must ensure that the production guests have priority in getting system resources. You can use the following z/VM SHARE values as starting values for the Linux guests:
    - Production guests: `SHARE 400 relative limitsoft`
    - Test guests: `SHARE 200 relative limitsoft`
    - Development guests: `SHARE 100 relative limitsoft`

Chapter 2: Scenarios

The reference architectures in this document address five typical customer scenarios. Each scenario builds on the last, increasing in complexity. Note that most of these scenarios concentrate on where the data is. We have chosen to concentrate on such scenarios because:
- A key strength of System z is its ability to be a highly-available database server.
- High Availability in distributed WebSphere applications is well documented already, but a need exists for architectures where WebSphere on z/Linux is using DB2 on z/OS.

For all scenarios, our goal is to provide a reference architecture that:
- Is rapidly scalable to support increases or decreases in business volume. Many times this can be accomplished simply by bringing more IFLs online to the existing architecture.
- Provides near-instantaneous failover, with almost no loss of user transactions.

Scenario: A non-WebSphere application

You have a critical application that runs on Linux on System z. This application does not use WebSphere or any database. The application may have been written by the customer, bought from an ISV, or be a server that is part of Linux, but it has no HA features itself.

Scenario: WebSphere with DB2 database on Linux

You have a critical WebSphere application that runs on Linux on System z. The primary database for this application is DB2 UDB also running on Linux on System z. The database files are on SCSI disks.

Scenario: WebSphere with Oracle database on Linux

You have a critical WebSphere application that runs on Linux on System z. The primary database for this application is Oracle also running on Linux on System z. The database files are on Extended Count Key Data (ECKD) disks.

Scenario: WebSphere with DB2 database on z/OS

You have a critical WebSphere application that runs on Linux on System z. The primary database for this application is DB2 running on z/OS. Some of the application logic runs as DB2 stored procedures.

Scenario: WebSphere with DB2 database on z/OS, in separate sites

You have several critical WebSphere applications that run on Linux on System z. The primary database for these applications is DB2 running on z/OS. You also need to ensure that if an entire data center is lost, another data center in a separate site can assume the work of the first data center.

Chapter 3: Reference Architecture: Non-WebSphere application

Scenario Being Solved

You have a key application that runs in Linux on System z in a Non-WebSphere environment and does not require a database. This could be a homegrown application.

Architecture Principles

This architecture is designed to follow these principles:

- Software is generally considered less reliable than hardware. The System z hardware contains redundant components, making its MTBF (Mean Time Between Failure) in the range of years. Because the System z hardware is so reliable, we allow the System z server to be a single point of failure in this architecture. We duplicate all the software environments (LPAR, VM, Firewalls, and Linux) so that none of them is a single point of failure.
- Any failure will not be noticeable to the user. The current transaction may fail, but subsequent transactions succeed. After any single failure, transactions continue at the same rate with no degradation in throughput or response time.
- The base architecture anticipates a low enough volume that it can be managed by a single server, but can scale if necessary to support increases and decreases in business volume.

Reference Architecture

The example scenario here will use an HTTP server as the example application. We will use a Service IP for availability purposes. A Service IP address is a single IP address by which the HTTP server is known to the outside world. In the event of a failover, this IP address must be reassigned to the new server. This choice offers the following benefits as an example:

- HTTP serving is a common application for Linux, yet is lightweight enough that another application could be easily substituted.
- The use of a Service IP not only illustrates solid availability, it also represents an additional required resource for the example. Even the simplest of web-serving arrangements requires at least one additional resource for dynamic content. Demonstrating the failover considerations afforded by a Service IP will provide a multi-resource example that can generalize to other examples requiring multiple resources.

Because this architecture is the simplest example in this document, we will demonstrate two different approaches to achieving high availability with an HTTP server and Service IP:

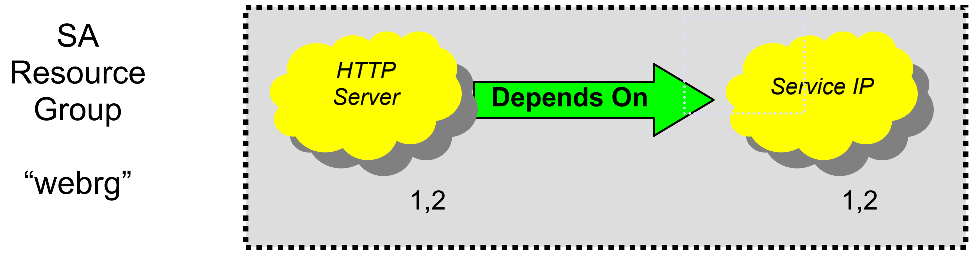
- **Using IBM Tivoli System Automation for Multiplatforms (SA):** SA allows the abstraction of resources in resource groups and has a powerful, policy-based mechanism for easily defining dependencies among resource-group elements. It is an appropriate choice for simplicity of service, as it is a fully-supported IBM product. Note that the SA example will demonstrate the use of a cold standby HTTP server and is thus not continuously available. In a failover event, there will be some downtime associated with bringing up the backup server.
- **Using Open Source packages, namely Linux-HA and Linux Virtual Server (LVS)**: This approach allows the creation of a load-balancing cluster of server nodes. The Service IP in this case is associated with an LVS Director instance, which sprays incoming requests over multiple HTTP Server instances. Because the server instances are clustered, one server going down will simply cause it to be fenced from the cluster. The LVS Director will not route any requests to it. Note that the LVS Director in this case is a single point of failure. We set up a second LVS Director as a standby, and use Linux-HA (also known as “Heartbeat”) between the two LVS Directors. The secondary LVS Director can be live, and thus failover times would be considered within an acceptable tolerance for continuous availability of the system. Note that this example uses a mix of open source applications from IBM and non-IBM projects and could thus represent additional service overhead. IBM Support for Linux-HA is available. IBM Support is not available for Linux Virtual Server. Note also that WebSphere Edge Components provide essentially the same function as LVS. The Edge Components Load Balancer can be configured with a hot standby. See “Chapter 4: Reference Architecture: WebSphere with DB2 Database on Linux” in this document for more information about WebSphere Edge Components.

First Example: Tivoli System Automation

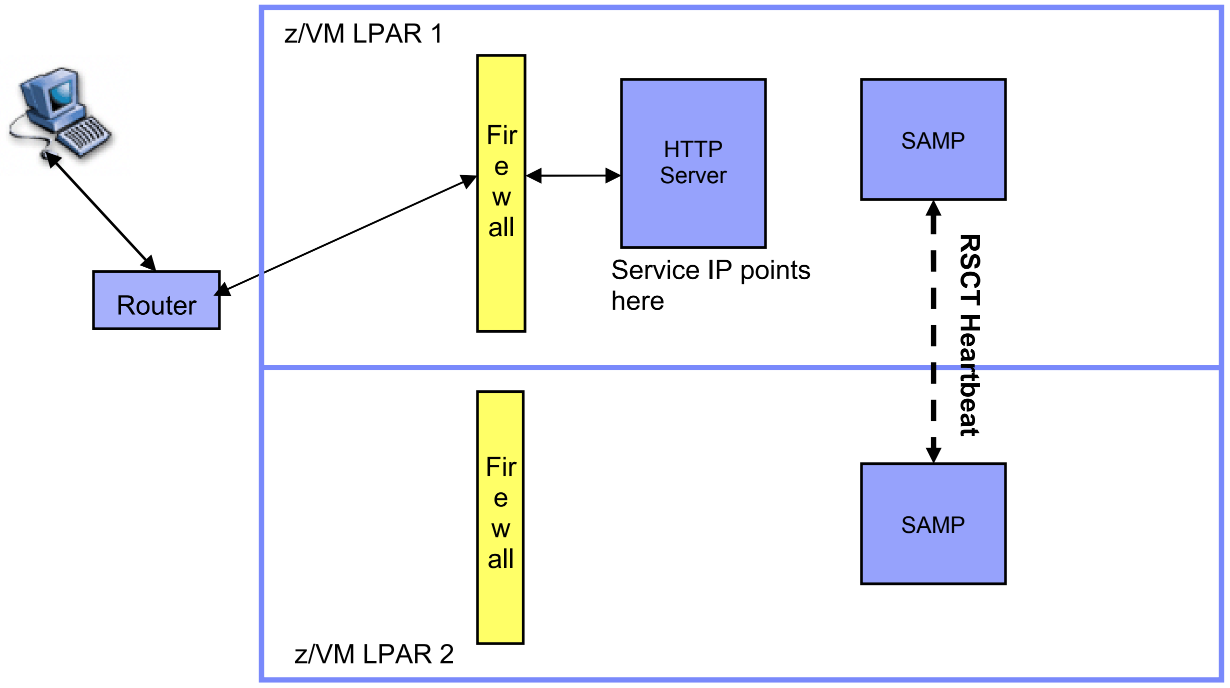
In this architecture, the HTTP Server and Service IP are defined as virtual resources in a SA resource group. The Service IP acts as a floating IP address. Its value remains fixed even if the Linux instance to which it points changes (such as through a failover). A Service IP is a single virtual IP address by which the currently-active HTTP Server is known to the router. The concept of a Service IP is not specific to SA, but SA can view a Service IP address as a virtual resource in a SA resource group. SA will handle the management of a Service IP by assigning the IP to the proper machine as needed. For example, the failure of the currently-active HTTP Server will cause SA to assign the Service IP to the assigned backup.

The HTTP Server depends on the Service IP for the address by which the router knows the server. The HTTP Server and Service IP instance are each known as “resources” in the SA Resource Group as shown in the diagram below. The 1,2 designation under each of the resources is a nodelist that designates the nodes on which the resource can exist. The strict “Depends On” relationship between the resources will cause SA to enforce collocation of the resources. SA also supports a “Depends on Any” relationship, in which resources in a resource group can be located on separate nodes.

The Depends On relationship also enforces a startup order between the two resources. A Service IP instance must point to a node before an HTTP Server can be started on that node. If either instance goes down, the Service IP will be assigned to the backup node, and then the HTTP Server will be started on the backup node. The nodelist as mentioned above refers to the two Linux instances that can run the resources managed by SA. These instances must be setup as members of an IBM Reliable Scalable Cluster Technology (RSCT) cluster. RSCT is an IBM software package that provides cluster monitoring and management services. SA installation includes installation of the RSCT package. RSCT provides the underlying resource monitoring (heartbeating) that SA uses to keep track of the state of nodes in the cluster. After the cluster is established, command-line commands are issued to manage the resources under SA.

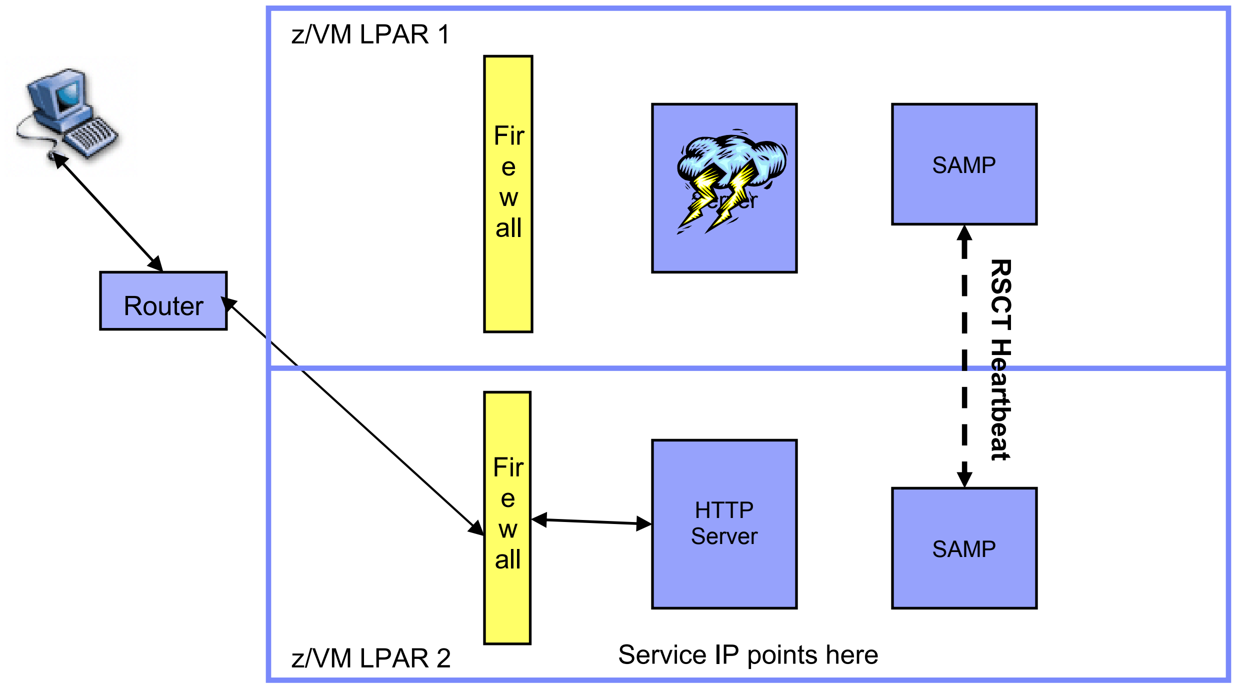


By virtue of this relationship SA will manage the IP address, associating it with the secondary server if anything goes wrong with the primary server, either the HTTP server application, the Linux O/S, or zVM. After the IP address has been assigned to the backup node, SA will start the HTTP server on that node. The initial state will be as shown in the following diagram:



Note that the RSCT/SA combination is strictly *managing* the HTTP Server and Service IP address. Neither RSCT nor SA is involved in the flow of processing actual HTTP services requests. If the RSCT/SA subsystem itself were to fail, the active HTTP Server would continue to function, albeit without the protection afforded by RSCT and SA.

In the event of a failure of the primary HTTP server, SA will assign the Service IP address to the secondary LPAR and bring up an HTTP Server on the LPAR. The resulting configuration is as shown:



Note that in this scenario, the standby server is cold. This could result in a service interruption as the server is started.

Flow of requests through this architecture

1. **Service IP.** Requests enter a router that is aware of a single IP address for the HTTP Server. The Service IP is associated with the active HTTP Server. In the event of a failure on the primary, SA will automatically assign the Service IP to the secondary.

2. **HTTP Server.** The HTTP server serves static content. The secondary is cold until started by SA.

Product Versions

- Any version of HTTP Server
- Tivoli System Automation for MultiPlatforms, V1.2
- Any version of z/VM

Planned Outages

This section discusses how each of the components can be taken down for software upgrades or any other planned outage. SA requires managed resources to be brought up and down under its control. In this example, all resources are together in a single group, and as a result, bringing either active resource down will cause SA to failover to the other LPAR. To bring a resource down under SA, use the `rgmbrreq` command.

As mentioned above, SA and RSCT are in place to manage the cluster and have no impact on the serving of HTTP content. Either or both of the SA or RSCT software can be brought down for maintenance without impacting the operation of the HTTP Server, except for the removal of RSCT/SA monitoring and failover “protection” for the HTTP Server.

What We Learned in Testing

When this architecture was set up and tested for planned and unplanned outages, we learned the following:

- Did the software failover as expected? Yes.
- Did users experience any outage time or transactions that they needed to retry? Yes
- Did users experience any permanent data loss? No
- How long did the failover take? (How long did users experience outages): Approximately 6 seconds to failover. Failover of the virtual ip is about 2 seconds, failover to http service is around 4 more seconds. Most of this time was to bring up a new HTTP server.

Architectural Decisions

Architectural decisions were made based on the following key criteria:

- High Availability
- Cost
- Simplicity

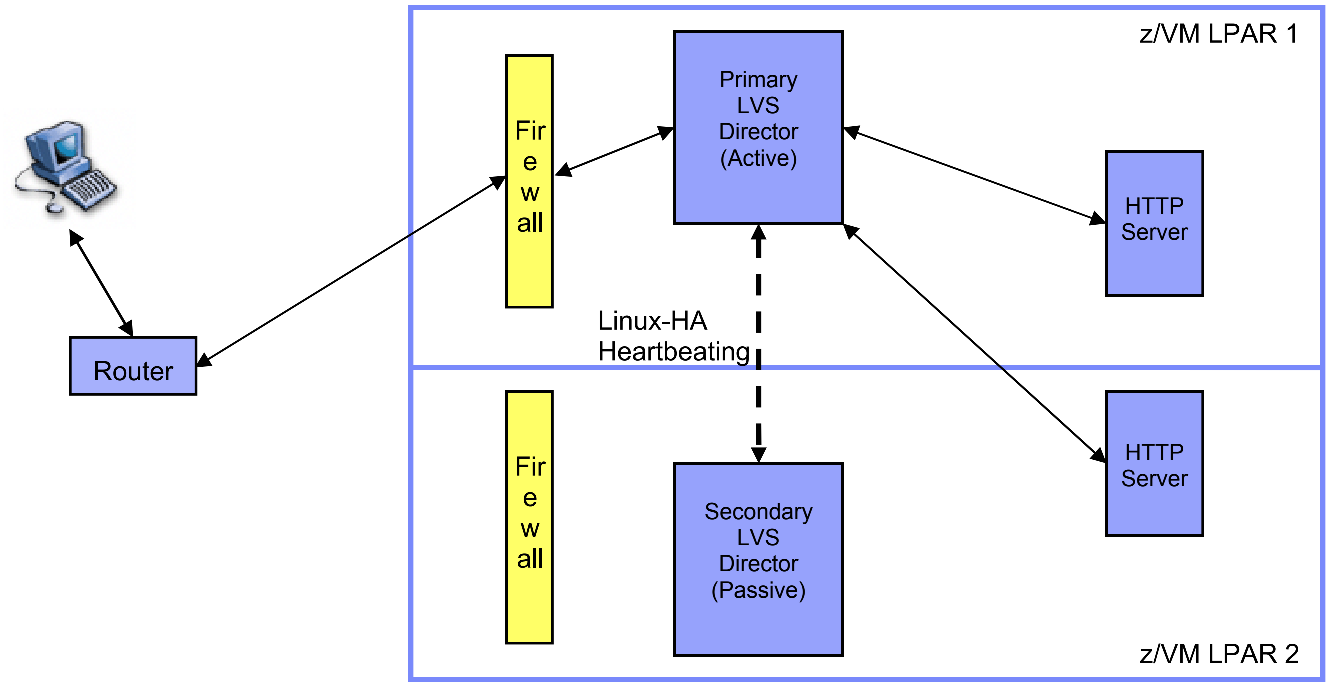
Architectural Decisions	
Decision Point	Pros/Cons
Use two separate System z servers to host the two LPARs.	This architecture can easily be run on two separate System z servers. The resource group can be simply failed over in its entirety to an LPAR on another physical server. If the two servers are in separate data centers, or in separate cities, then this solution will cover the “Disaster Recovery” situation, where one data center is lost and the other can take over the workload immediately.
Use of a Service IP Address	The use of a Service IP was chosen for two reasons. First, it abstracts the HTTP Server instances in the same way as Virtual IP Addressing (VIPA), resulting in a more simple failover. Second, it allows the illustration of how to handle multiple dependent resources in a group. It is expected that user applications will typically be comprised of multiple software components. The example, as illustrated, will be easier to draw upon in such situations.
Maintaining the “Depends On” relationship between resources	The HTTP Server is dependent on the Service IP instance for obvious reasons. The choice of “Depends On” results in forced collocation. “Depends On Any” would allow for the HTTP Server to exist on a separate LPAR than the Service IP instance.

As mentioned above, SA as an architectural choice affords a robust support structure. While IBM support is available for Linux-HA, it is not available for Linux Virtual Server. IBM has found that System z customers in general prefer supported software. In addition, while this example is simple, SA scales well and so is suitable for managing availability in very complex environments with intertwined dependencies.

Second Example: Linux Virtual Server and Linux-HA

For the open source design, we rely on Linux Virtual Server (LVS). LVS requires a single director node as well as any number of application instance, or cluster, nodes. All requests for services come through the director and are assigned to cluster nodes to process the actual workload (for example, by round robin). High availability among the cluster nodes is implicit, since the failure of one of those nodes will cause it to be removed from consideration by the LVS director, and the remaining cluster nodes will simply assume responsibility for the workload.

For the open source design, we use a passive secondary LVS director node. Linux-HA manages the director nodes, including automated failover. This failover capability eliminates the single point of failure.



In this architecture, the single IP address is associated with and managed by the active LVS Director. The HTTP Server instances are both hot, which allows for load balancing of requests across them. If one of the HTTP server nodes goes down, the LVS Director will remove it from the list of active servers. LVS and Heartbeat can also work together to monitor the HTTP servers using Ldirector. An excellent description of how to setup LVS and Heartbeat in this configuration can be found at http://mail1.cula.net/cluster/index.html. See also "*System z Platform Test Report for z/OS and Linux Virtual Servers*" for more details.

Flow of requests through this architecture

1. **LVS Director.** Requests enter a router that is aware of a single IP address for the HTTP Server. The IP address is associated with the LVS Director. The LVS Director will route requests to the HTTP Servers (for example, by round robin).

2. **HTTP Server.** The HTTP server serves static content. Should any of the HTTP servers fail, the failing server will no longer receive requests from the LVS Director. If Ldirector is used as described above, than HTTP Servers that come back online will be automatically placed back into the cluster of active servers by the LVS Director.

Product Versions

- Any version of HTTP Server
- Linux-HA (Heartbeat) version 2.0, which includes support for many new features

- Any version of LVS
- Any version of Ldirector compatible with the Linux-HA and LVS versions chosen (Ldirector is available through the Linux-HA site.)
- Any version of LVS
- Ultra Monkey version 3 (if Ultra Monkey is to be used)
- Any version of z/VM

Note that the implementation and configuration details provided at http://mail1.cula.net/cluster/index.html also describe the specific software packages to download. The versions are workable but rather outdated. IBM recommends that this description be used as a guideline but that more recent versions of all packages be used.

Planned Outages

This section discusses how each of the components can be taken down for software upgrades or any other planned outage. There are two sets of redundant elements, namely the HTTP servers and the LVS Directors. In either case, the system is set up such that a single element of either set can be brought down manually without impacting the system. In the case of the HTTP servers, either server could be brought down and LVS would simply stop sending requests to that server. Using Ldirector, the server would be returned to the cluster upon startup.

The LVS Directors are slightly different since only one is in active state. Bringing down the secondary is trivial because it is not active. Bringing down the primary will cause Heartbeat to failover to the secondary.

Note that multiple backup LVS Directors can be configured. If only a single backup Director is deployed, bringing down that backup will make the primary a single point of failure. A description of the management of redundant LVS Directors using Heartbeat and Ldirector can be found at http://www.austintek.com/LVS/LVS-HOWTO/HOWTO/LVS-HOWTO.failover.html. The UltraMonkey project is an Open Source project that implements such management. See http://www.ultramonkey.org/ for more information about UltraMonkey.

What we Learned in Testing

When this architecture was set up and tested for planned and unplanned outages, we learned the following:

- Did the software failover as expected? Yes.
- Did users experience any outage time or transactions that they needed to retry? No
- Did users experience any permanent data loss? No
- How long did the failover take? (How long did users experience outages): No user outages were seen. Failover happened in approximately 1 second. As long as this remains below the TCP/IP timeout value, no transactions are lost.

Architectural Decisions

Architectural decisions were made based on the following key criteria:

- High Availability
- Cost
- Simplicity

Architectural Decisions	
Decision Point	Pros/Cons

Use of Linux Virtual Server (LVS)	LVS is a standard, robust open-source package for load balancing and high availability in Linux clusters.
Use of Multiple LVS Directors	All requests must go through the LVS Director, making it a single point of failure without redundancy. Simple heartbeating between the primary and secondary LVS Directors allows for a highly-available configuration.
Use of Linux-HA/Heartbeat	Linux-HA is a standard, robust open-source package for heartbeating in Linux clusters.

The choice of LVS and Linux-HA provides the benefits of server load balancing. Clusters of load-balanced servers afford better utilization and continuous server availability.

Chapter 4: Reference Architecture: WebSphere with DB2 database on Linux

Scenario Being Solved

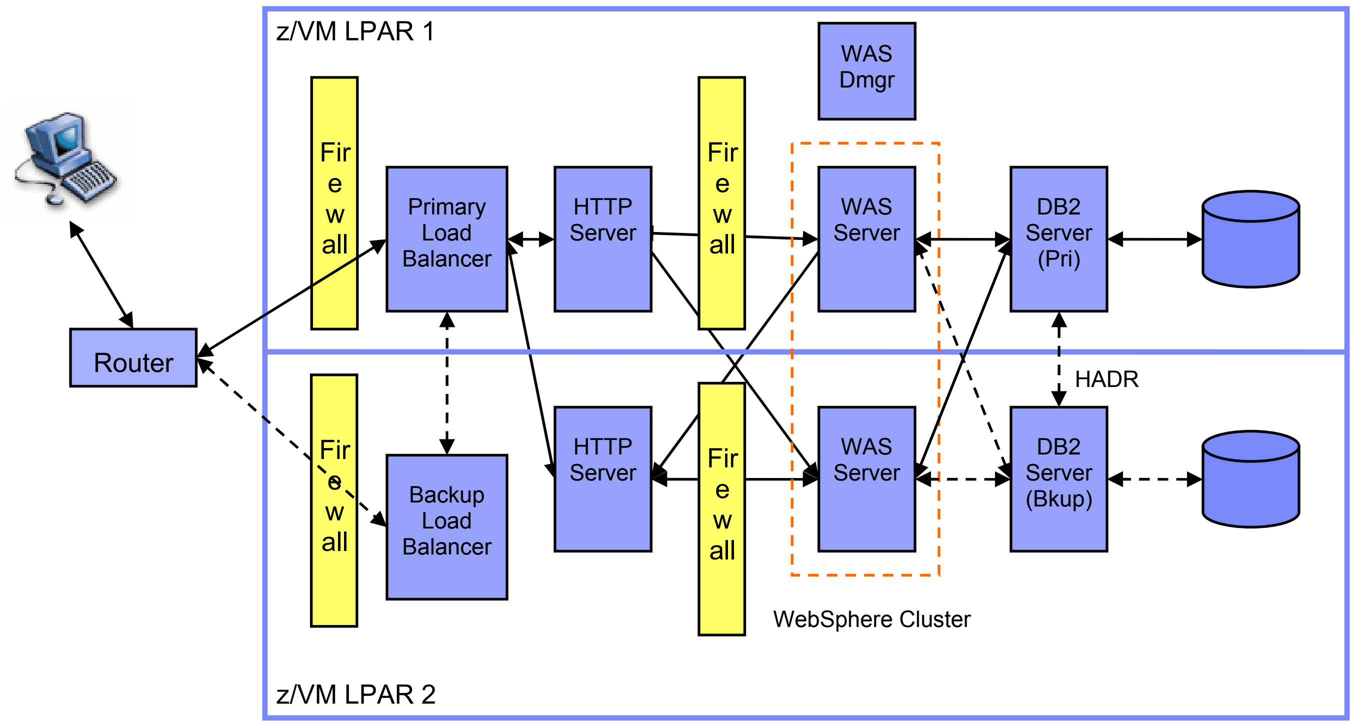
You have a key WebSphere application that runs in Linux on System z. The primary database for this application is DB2 UDB, also running on Linux on System z. The database files are on ECKD disks.

Architecture Principles

This architecture is designed to follow these principles:

- Software is generally considered less reliable than hardware. The System z hardware contains all redundant components, making its Mean Time Between Failure (MTBF) in the range of decades. Because the System z hardware is so reliable, we allow the System z server to be a single point of failure in this architecture. We duplicate all of the software environments (LPAR, VM, Firewalls, Linux, WebSphere, DB2) so that none of them is a single point of failure.
- No failure should be noticeable to the end user. The current transaction may fail, but subsequent transactions succeed. After any single failure, transactions continue at the same rate with no degradation in throughput or response time.
- The architecture must be rapidly scalable to support increases and decreases in business volume.

Reference Architecture



Each box represents a virtual Linux server running as a guest of z/VM, in two z/VM LPARs.

In this architecture, all software servers are duplicated on two Logical Partitions (LPARs) on the same System z server. Outer and inner firewalls form a DMZ for the Edge servers and HTTP servers.

Flow of requests through this architecture

1. **Communications within the LPAR.** All communications between Linux guests within a z/VM LPAR are completed through a z/VM Virtual Switch (vswitch). The vswitch is a fast and scalable communication infrastructure. We recommend setting up one vswitch with two vswitch controllers (VM user ids) in each z/VM LPAR. (Note that z/VM 5.2 comes with two VSWITCH controllers defined. We recommend that you XAUTOLOG the two of them: DTCVSW1 and DTCVSW2.)

   Each vswitch should be set up to use multiple real devices, so that if one OSA fails, the VSWITCH will fail over to the other.

2. **Load Balancer.** Requests enter a router that is connected to the Virtual IP Address (VIPA) of the Primary Load Balancer. Should this load balancer fail, the backup load balancer will detect the failure and take over. The router will detect this failure and route requests to the backup load balancer.

3. **HTTP Server.** The load balancer sprays requests between the two HTTP servers. Should one of the HTTP servers fail, the load balancer detects this and will not route requests to it. The HTTP server serves static pages. It also routes WebSphere requests via the WebSphere plugin to the two WebSphere servers in the cluster. Should one of the WebSphere servers fail, the plugin will detect this and not route requests to it.

4. **WebSphere.** The application is deployed to a WebSphere cluster consisting of two nodes. WebSphere manages the deployment of the application onto the nodes of the cluster and can upgrade the application on the fly.

   User session data is replicated among the cluster members so that if a cluster member fails the transaction can be continued on the other cluster member. We recommend configuring WebSphere to replicate session data and hold it in memory in the cluster members. This option performs well and can scale well, and is more simple to configure than using a database to hold session data.

   In flight Two Phase Commit (2PC) transactions can be recovered by the WebSphere HA manager. The HAManager is a new feature of WebSphere v6. It enhances the availability of WebSphere singleton services like transaction services or JMS message services. It runs as a service within each application server process that monitors the health of WebSphere clusters. In the event of a server failure, the HAManager will failover the singleton service and recover any in-flight transactions. In order to do this the transaction logs written by each application server must be on network-attached storage or a storage SAN so that they are readable to the remaining cluster members. Note that this setup is optional and is not depicted in our reference architecture. When the HAmanager coordinator detects that an application server is down, it can initiate recovery of in-flight transactions from the transaction logs. This recovery will release any locks held in the database.

   The following steps are required to accomplish this setup:
   - Make the transaction logs sharable by all members of the cluster. By default these are located in the <WASinstallroot>\profiles\<profilename>\tranlog\<cellname>\<nodename>\<servername>\transaction directory, but should be configured for another directory that you will make sharable.
   - After you configure the logs, the only other setup is to “Enable high availability for persistent services“ by checking the box by that name in the ServerCluster.

   Refer to Redbook SG24-6392: WebSphere Application Server V6 Scalability and Performance Handbook, section 9.7 “Transaction Manager High Availability” for details about how to set up the HAmanager in this way.

5. **DB2 Client (JDBC).** WebSphere runs the application and sends DB2 data requests to the Primary DB2 Server. HADR communicates to the DB2 clients (the JDBC driver in our case) when they first connect to DB2 to inform them of the address of the backup server. If any communication to the primary DB2 server fails, the clients automatically route requests to the backup server. This is the Automatic Client Reroute feature. Configure the WebSphere connection pool settings in the DB2 data source to use purge.policy=pool. This will cause WebSphere to empty out the entire pool upon a single connection failure. Without this option, WAS will hand out stale connections until it purges the pool of all connections.

6. **DB2.** The DB2 HADR (High Availability Disaster Recovery) feature is used to provide failover in case of a DB2 failure. HADR uses two DB2 servers and two databases to mirror the data from the primary database to the backup. Tivoli System Automation running on both DB2 Servers automatically detects a failure of the primary and issues commands on the backup for the DB2 there to become the primary. Since it has been mirroring all data from the primary, the backup does not need to do any database recovery before becoming primary. So the database takeover is accomplished as fast as can be detected by TSA.

   Details on the DB2 failover setup:

- DB2 provides the TSA MP scripts to automate the HADR takeover. Install these scripts into each TSA server. See the "Automating DB2 HADR Failover on Linux using Tivoli System Automation" whitepaper for more information.
- The WebSphere application needs to look for a specific SQL return code that indicates that the primary DB2 server has failed. This return code means that any transactions prior to COMMIT have been rolled back. The application needs to reissue the previous transactions.
- Use the DB2 "update alternate server for database" command at each DB2 node to identify the other DB2 server. With this setup, all connecting clients will become aware of the backup server after connecting to the primary. This is only necessary for type2 drivers.
- HADR is included in DB2 UDB 8.2 Enterprise Server Edition.
- HADR ACR (automatic client reroute) works with type 2 and type 4 JDBC clients, but for XA transactions the type 4 client is required.
  On WebSphere Linux, the XA implementation for two-phase commit is the only one available for WebSphere managed transactions. As a result, resources need to be specifically declared as XA (that is, XADataSource) if you want to participate in 2PC. The type 4 JDBC client is needed for XA-type transactions that need automatic client reroute. (XA is supported for both type2 and type4 clients, but XA and ACR only works for type4.)

7. **Disk Multipathing**. The database volumes are configured for multipathing, so that if one path fails, access to the device is maintained through the surviving paths. For ECKD DASD devices accessed over FICON or ESCON channels, multipathing is handled invisibly to the Linux operating system. A single device is presented to the operating system on which to do I/O operations. Multipathing happens automatically and is handled by the System z I/O subsystem. All that is required is for multiple paths to be defined to the device in the active I/O Definition File (IODF), and for those paths to be online. The complexity of choosing among the multiple paths is hidden from the Linux OS.

Product Versions

This architecture requires the following software versions:

- WebSphere Network Deployment V6.0 or newer. This architecture can be done with WebSphere v5, but it lacks some of the HA features of WebSphere v6. For these reasons we recommend v6.
- DB2 UDB 8.2 is needed for the HADR function. You also need the JDBC driver that comes with 8.2, to get the Automatic Client Reroute feature.
- Tivoli System Automation for MultiPlatforms, V1.2. This is included with DB2 8.2 at no cost for use with DB2.
- z/VM 4.4 or above, to get vswitch.

Planned Outages

This section describes how each of the components can be taken down for software upgrades or any other planned outage.

Planned Outages	
Component	Procedure
Load Balancer	1. Stop the network dispatcher component on the server you need to upgrade. The backup detects that the primary is stopped and becomes the primary. The router detects that the primary is down and routes requests to the backup. 2. After the primary has been upgraded, start the Network Dispatcher component and make it primary again. 3. Go to the backup server and apply the upgrade there.

HTTP Server	1. Stop the HTTP Server on the server you need to upgrade. The load balancer detects that this HTTP server is not responding and will stop sending requests there. 2. After the HTTP Server has been upgraded, restart the HTTP Server. As soon as it is started, the load balancer detects that it is available and starts sending requests to it. 3. Go to the other HTTP server and repeat steps 1 and 2.
WebSphere Application Server	To upgrade the application running in the cluster, simply perform the "Rolling upgrade" function from the WAS ND administrative panels. This drains each request queue, stops the application server, upgrades the application, and starts the app server again. WAS will make sure the upgraded app server is up and handling requests before upgrading the next cluster member. If session replication in enabled between the cluster members, then this method will cause no loss of transactions. To apply service to WebSphere 1. From the WAS ND administrative panels, stop one application server in the cluster. This process ensures that no in-flight transactions are interrupted by the stop by first notifying the WebSphere plug-in running in the HTTP server that WAS is down and to stop sending new requests. Then WebSphere finishes bringing down the application server. 2. Upgrade WAS or do whatever other planned outage work needs to be done on that server. 3. Start the WAS application server again. Wait until the app server is again handling work. 4. Repeat this process on the other application server in the cluster.
DB2	1. On the backup DB2 server, stop HADR mirroring. 2. Apply the upgrades. 3. Start HADR again and allow the DB2 server to catch up with the changes made on the primary. 4. After this process is complete, tell the backup to become the primary, using the TAKEOVER command. 5. Repeat this process on the primary DB2 server.

What we Learned in Testing

When this architecture was set up and tested for planned and unplanned outages, we learned the following:

- Did the software failover as expected? Yes.
- Did users experience any outage time or transactions that they needed to retry? Yes
- Did users experience any permanent data loss? No
- How long did the failover take? (How long did users experience outages): Approximately 1 minute for HADR to failover and WebSphere to stop sending requests to the failed DB2 server and redirect to the alternate.

Architectural Decisions

Architectural decisions were made based on the following key criteria:

- High Availability
- Cost
- Simplicity

Architectural Decisions	
Decision Point	Pros/Cons
Use two separate System z servers to host the two LPARs.	This architecture can easily be run on two separate System z servers. All communication between cluster members is through TCP/IP, and the DB2 HADR mirroring can be done in asynchronous mode so that network delays between System z systems will not be a problem. This removes the System z hardware as a single point of failure. However, it significantly increases the cost of the solution. There must be the same number of IFLs available on the second System z as there is on the first, since either must be able to run 100% of the workload without degrading response time. So the number of IFLs needed to run the workload is double the number needed to run the workload on two LPARs who *share* the same System z server. If the two servers are in separate data centers, or in separate cities, then this solution will cover the "Disaster Recovery" situation, where one data center is lost and the other can take over the workload immediately. See the architecture for Scenario 5 for more information. Another con is that encryption of all of the data traffic is now required. When all of the data traffic was on the same CEC, there was no need for encryption because no communication ever left the System z machine.
Use WebSphere memory replication to save user session data	WebSphere gives you three levels of HTTP session persistence: • No persistence • Harden session data to a database • Keep session data in memory in all cluster members. Hardening session data to a database is the most robust and scalable solution, but also requires that the database be HA-enabled itself. Memory replication means that session data from each cluster member is available to all of the other cluster members, but not hardened to disk. The session data is kept in JVM memory. This means that session data is lost only if the entire cluster fails. For moderate workloads, memory replication performs well but uses more memory than database hardening. But as WebSphere scales to very large numbers of users and large session data, memory replication can perform worse than hardening to disk because of the following: • WebSphere spends lots of CPU time synching the session data on all the app servers. • The amount of JVM memory needed to store session data increases, leaving less for the application, and requiring larger JVM heaps. WebSphere offers several ways to set up memory replication so that either of the

	following is true: • All of the session data is in memory in each application server in the cluster. • The session data is held in memory in a separate application server that is dedicated to this task. This configuration can work well with very high transaction rates or large amounts of session data, because all of the session data does not have to be replicated in each application server. However, this separate server becomes a new single point of failure. WebSphere v6.0.2 increased the performance of memory-to-memory session replication to equal that of database replication. For this and the reasons above we recommend using TBW (time-based write) memory replication of session data for an HA configuration. Refer to the June 2005 edition of "WebSphere Technical Journal", available at http://www.ibm.com/developerworks/websphere/techjournal/0506_col_alcott/0506_col_alcott.html , and the book "*WebSphere Deployment and Advanced Configurations*" for an excellent article on this subject.
Where to run the WebSphere Deployment Manager	In WebSphere v6, the deployment manager is only used for deployment and configuration, and does not perform any HA monitoring of the cluster like it did in v5. So we feel that it is not required to HA-enable it. Therefore we recommend you run the Dmgr in a separate Linux guest on either of the Linux LPARs.
Use TSA alone (without DB2 HADR) to provide failover for DB2	Tivoli System Automation can be set up to manage the availability of the DB2 instance in a 2-node cluster. A whitepaper and TSA scripts can be downloaded for this solution from http://www.ibm.com/software/tivoli/products/sys-auto-linux/downloads.html. In the event of a database failure, TSA activates the backup DB2 instance (which is an active but idle Linux guest with DB2 already started) and triggers the DB2 recovery mechanisms. DB2 provides the TSA scripts to do this. The DB2 transaction log is used to replay committed transactions and undo in-flight transactions, in order to bring the on-disk image to a consistent state in the event of a crash. Because the DB2 server node and the spare node are both connected to the same disk subsystem, when DB2 on the spare node is activated during failover, it can simply start the recovery process using the database storage, transaction logs, and configuration information written to disk by the original DB2 node. No transfer of logs or database images is required between the failing system and the spare system, enabling faster recovery time. An issue with this is the speed of recovery. Using HADR, failover from the primary to the secondary DB2 server takes only a few seconds, and client requests will not time out due to this delay. Using TSA alone to kickoff DB2 recovery can take up to 60 seconds to start and recover DB2. HADR can also easily handle rolling upgrades to DB2. Simply upgrade the backup server, and then use the DB2 Control center to switch the backup to become the primary, and upgrade the other server.
Use SCSI devices instead of ECKD.	SCSI devices accessed over FCP paths can be used instead. The FCP/SCSI approach is described in Chapter 5. Each is used once in this document in order to articulate the differences in configuring multipathing and device sharing for each.

Key problem areas

Non-functional requirements:

- *Data integrity.* No data loss is permitted, even in disaster (site fail-over) scenario.
- *Security*. The Architecture can be used for systems with stringent security requirements. It provides layered defenses against internal and external threats.
- *Performance and Scalability.* This Architecture supports systems with medium throughput requirements and allows for additions to scale to greater volumes.

Solution estimating guidelines

Use the “WebSphere on Linux on System z Sizing” process to estimate the number of IFLs required for HTTP, WebSphere, and DB2.

Chapter 5: Reference Architecture: WebSphere with Oracle database on Linux

Scenario Being Solved

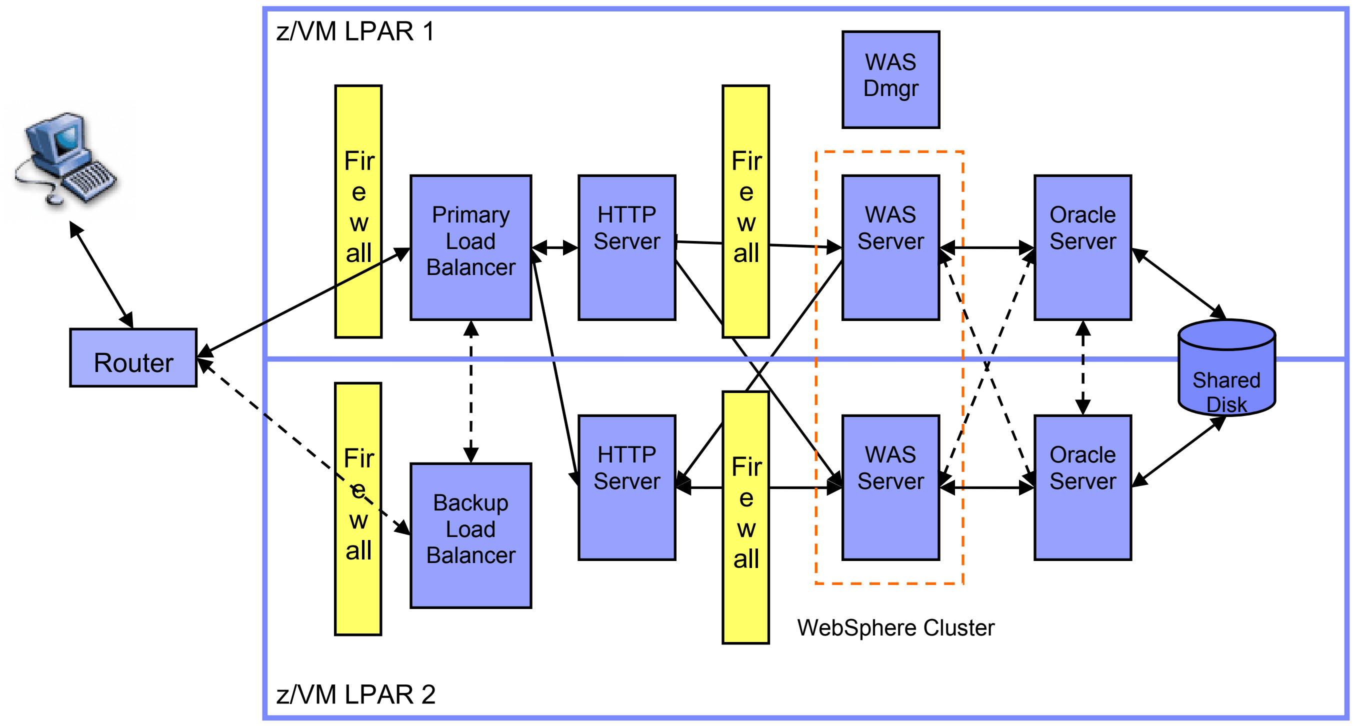
You have a key WebSphere application that runs in Linux on System z. The primary database for this application is Oracle, also running on Linux on System z. The database files are on SCSI disks.

Architecture Principles

This architecture is designed to follow these principles:

- Software is generally considered less reliable than hardware. The System z hardware contains all redundant components, making its MTBF in the range of years. Because the System z hardware is so reliable, we allow the System z server to be a single point of failure in this architecture. We duplicate all of the software environments (LPAR, VM, Firewalls, Linux, WebSphere, Oracle) so that none of them is a single point of failure.
- Failure should not be noticeable to the end user. The current transaction may fail, but subsequent transactions succeed. After any single failure, transactions continue at the same rate with no degradation in throughput or response time.
- The architecture must be rapidly scalable to support increases and decreases in business volume.

Reference Architecture



Each box represents a virtual Linux server running as a guest of z/VM, in two z/VM LPARs.

In this architecture, all software servers are duplicated on two Logical Partitions (LPARs) on the same System z server. Outer and inner firewalls form a DMZ for the Load Balancers and HTTP servers.

Flow of requests through this architecture

1. **Load Balancer**. The router and Load Balancer are configured the same as it is described on page 18.
2. **HTTP Server.** The HTTP server is configured the same as it is described on page 18.
3. **WebSphere.** WebSphere is configured the same as it is described on page 19.

4. **Oracle Client (JDBC).** WebSphere runs the application and sends requests to the Oracle database server (Oracle calls this an "instance"). Communication with Oracle is via the Oracle Thin Client, which is a type 4 JDBC driver. It uses a self contained, light weight version of SQL*Net to communicate over TCP/IP with the Oracle database instance. The WebSphere datasource lists the instances that make up the Oracle RAC cluster. If the primary instance is unavailable, the driver will use an alternate instance. The thin client should be defined to use "session" failover, meaning that if a connection is lost from the client to the instance, a new session is automatically created with another instance (session failover does not attempt to recover selects.) Since the thin client handles the reestablishment of connections to an alternate instance, no purging of the WebSphere connection pool is necessary.

5. **Oracle.** The servers are configured using Oracle Real Application Cluster (RAC) in an active/active configuration with two instances. This means that each instance is active and updates a common database. Should one of the Oracle instances fail, the in-flight transactions are lost, but the other instance in the RAC can receive all JDBC requests immediately. Configuration data is in the tnsnames.ora file on each instance.

6. **Disk Multipathing**. The database volumes are configured for multipathing, so that if one path fails, access to the device is maintained through the surviving paths. Unlike FICON-attached ECKD DASD devices, when SCSI DASD devices are accessed over System z FCP channels, each path to each LUN appears to the operating system as a different device. For example, if there are four paths to five LUNs, the Linux system sees 20 SCSI devices. This means that there must be another layer of code between the Linux filesystem layer and the subsystem. This extra layer handles all of the coordination between the raw paths and the higher level subsystem. On Red Hat Linux, this extra layer is handled by software RAID and mdadm. On SUSE Linux it is handled by LVM (SLES 8) or the Device Mapper subsystem in the 2.6 kernel in conjunction with EVMS (SLES 9). So, unlike the FICON/ECKD case, FCP/SCSI multipathing *is* managed by the Linux OS. There is no global multipathing scheduler that works across the entire System z system for FCP.

   The mdadm approach used by Red Hat currently does not balance loads among available paths to LUNs. Rather, it uses a primary path to the LUN, then fails over to a secondary path if there are any problems with the primary path. When an active path fails, the md subsystem detects the failure, marks the path failed, then makes a secondary path active. If the failed path comes back, the md subsystem recognizes this and brings it back as a secondary path.

   For information about configuring FCP multipathing for Red Hat or SUSE systems, see the Redbooks SG24-6344, *Linux for System z: Fibre Channel Protocol Implementation Guide* and SG24-6694, *Linux for IBM System z9 and System z*.

Product Versions

This architecture requires the following software versions:

- WebSphere Network Deployment V6.0 or newer. This architecture can be done with WebSphere v5 but lacks some of the HA features of WebSphere v6. For these reasons we recommend v6.
- Oracle Database 10g Release 1 (10.1.0.3) with RAC feature. Oracle thin client JDBC type 4 driver.

Planned Outages

This section discusses how each of the components can be taken down for software upgrades or any other planned outage.

Planned Outages	
Component	Procedure
Load Balancer	Same as in previous architecture.
HTTP Server	Same as in previous architecture.
WebSphere Application Server	Same as in previous architecture.
Oracle	Use the `SHUTDOWN TRANSACTIONAL` command with the `LOCAL` option to make an instance (RAC Server) shutdown after all active transactions on the instance have either committed or rolled back. Transactions on other instances do not block this operation.

What we Learned in Testing

When this architecture was set up and tested for planned and unplanned outages, we learned the following:

- Did the software failover as expected? Yes.

- Did users experience any outage time or transactions that they needed to retry? Yes
- Did users experience any permanent data loss? No
- How long did the failover take? (How long did users experience outages): Approximately 2 minutes for RAC to failover and WebSphere to stop sending requests to the failed Oracle instance and redirect to the alternate. The active/passive configuration of RAC may fail over faster.
- Problems were experienced during failback (when the primary instance was restarted and the alternate stopped). Transaction throughput dropped significantly or stopped. We assumed this was a configuration error that we could not find.

Architectural Decisions

Architectural decisions were made based on the following key criteria:
- High Availability
- Cost
- Simplicity

Since this architecture is so similar to the previous one, we have not duplicated all of the discussion of the architectural decisions here. Only those that are unique to this architecture are described below.

Architectural Decisions	
Decision Point	Pros/Cons
Using Oracle DataGuard	While RAC provides a cluster of Oracle instances all sharing one database, DataGuard provides for mirroring of the database itself using a primary and standby database. To duplicate both the instances and the database requires both RAC and DataGuard. Using DataGuard with “transaction integrity”, if there is a disk failure then the standby database will be current to the last committed transaction. If there is a disk failure when using RAC only, then a database recovery is needed (Oracle 10g keeps recent transactions in a flash-back area making that kind of recovery very quick.) Dataguard is a good solution to provide high availability for RAC clusters on separate System z servers. The dataflow between the clusters is minimal because redo log shipping is used. There are several choices as to how current to keep the backup cluster, including transactional integrity.
Using an Active/Passive configuration instead of Active/Active	There is a high CPU cost for an active/active RAC configuration. Using active/passive RAC uses less CPU, and allows faster failovers, but does not support load balancing since only one RAC instance is active.
Use ECKD devices instead of SCSI	ECKD devices accessed over FICON or ESCON paths can be used instead. The ECKD approach is described in the previous architecture. Each is used once in this document in order to articulate the differences in configuring multipathing and device sharing for each.

Key problem areas

Non-functional requirements are the following:
- *Data integrity.* No data loss is permitted, even in a disaster (site fail-over) scenario.

- *Security*. The architecture can be used for systems with stringent security requirements. It provides layered defenses against internal and external threats.
- *Performance and scalability.* This Architecture supports systems with medium throughput requirements and allows for additions to scale to greater volumes.
- *Cost.* Oracle RAC is quite costly, but provides good function.

Solution estimating guidelines

Use the “WebSphere on Linux on System z Sizing” process to estimate the number of IFLs required for HTTP, WebSphere, and Oracle. The sizing actually uses DB2, but the cycles for DB2 and Oracle are similar enough that the sizing will be valid.

Chapter 6: Reference Architecture: WebSphere with DB2 database on z/OS

Scenario Being Solved

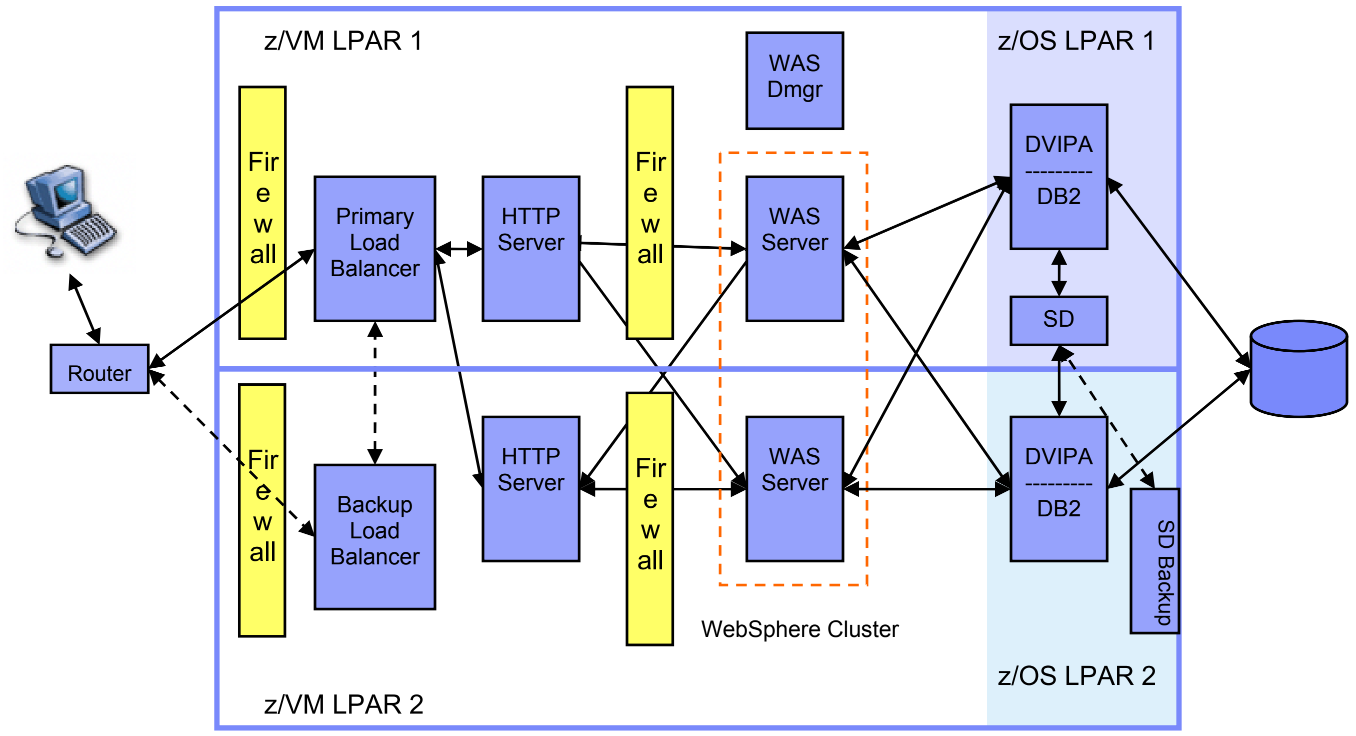
You have a key WebSphere application that runs in Linux on System z. The primary database for this application is a DB2 data sharing group running in a Parallel Sysplex on z/OS.

Architecture Principles

This architecture is designed to follow these principles:

- Software is generally considered less reliable than hardware. The System z hardware contains all redundant components, making its MTBF in the range of years. Because the System z hardware is so reliable, we allow the System z server to be a single point of failure in this architecture. We duplicate all of the software environments (VM, Firewalls, Linux, WebSphere, DB2) so that none of them is a single point of failure.
- No failure should be noticeable to the end user. The current transaction may fail, but subsequent transactions succeed. After any single failure, transactions continue at the same rate with no degradation in throughput or response time.
- The architecture must be rapidly scalable to support increases and decreases in business volume.
- The architecture should leverage the high availability capabilities and features of z/OS Parallel Sysplex.

Reference Architecture



Each box on the left represents a virtual Linux server running as a guest of z/VM, in two z/VM LPARs. On the right are two z/OS LPARs. The boxes within them represent z/OS processes.

This architecture introduces the use of a DB2 z/OS data sharing group as the data store. The data sharing function of DB2 z/OS enables applications that run on different DB2 subsystems to read and write the same data concurrently. DB2 subsystems that share data must belong to a DB2 data sharing group, which runs in a Parallel Sysplex cluster. A data sharing group is a collection of one or more DB2 subsystems that access shared DB2 data. Each DB2 subsystem that belongs to a particular data sharing group is a *member* of that group. All members of a data sharing group use the same shared DB2 catalog and directory, share user data, and behave as a single logical server with the benefit of higher scalability and availability. The maximum number of members in a data sharing group is 32.

Flow of requests through this architecture

1. **Load Balancer**. The router and Load Balancer are configured the same as described on page 18.
2. **HTTP Server.** The HTTP server is configured the same as described on page 18.
3. **WebSphere.** WebSphere is configured the same as described on page 19.

4. **JDBC Type 4 Driver.** Each WAS server is configured to use a pure Java driver for connectivity to DB2 UDB on z/OS, called the DB2 Universal JDBC Driver type 4 (JDBC T4). JDBC T4 is sysplex-aware and can intelligently route workload across a DB2 data sharing group in a Parallel Sysplex.

   On z/OS, Sysplex Distributor provides an initial contact single cluster IP address (known as a group Dynamic VIPA) for the data sharing group, which has built-in redundancy across network adapters and z/OS images. Each DB2 data-sharing group member is also addressable by its own Dynamic VIPA (Virtual IP Address) to insulate it from outages of any individual network adapter, and from potential restarts of the DB2 group member on other z/OS images in the parallel sysplex. z/OS WLM (Workload Manager) works with DB2 z/OS and JDBC T4 to direct subsequent traffic to the DB2 group member with the most available capacity on a transaction-by-transaction basis.

5. **DB2**. Should a DB2 z/OS group member fail, in-flight work to that DB2 will fail and be backed out, but subsequent transactions will be automatically rerouted to surviving DB2 group members. z/OS ARM (Automatic Restart Manager) can automatically restart a failed DB2, either in-place if its host z/OS is still available, or on another z/OS system in the sysplex if its original z/OS host is no longer active (based upon whatever restart policy the user has previously specified). If an entire z/OS host has either failed or appears hung, z/OS SFM (Sysplex Failure Management) can perform system isolation to cleanly remove the z/OS system from the sysplex, either automatically or based upon operator request (based on whatever policy the user has previously specified).

More detail on these components

Let's look at each of the components in this architecture in more detail. The focus here is on the flow from Linux on System z to DB2 on z/OS. A comprehensive view of z/OS Parallel Sysplex high availability configurations and options is beyond the scope of this document, though a few aspects of z/OS Parallel Sysplex high availability that are particularly relevant to this discussion are covered.

DB2 UDB for z/OS: Each DB2 member in the Parallel Sysplex interfaces with z/OS WLM to obtain a prioritized list of DB2 data sharing group members based on their availability and useable capacity. After a sysplex-aware JDBC T4 client has made initial contact with DB2 UDB for z/OS, DB2 returns this list to the JDBC T4 client.

JDBC T4: JDBC T4 is configured to be sysplex-aware, and to use connection concentrator in addition to the connection pooling available within WebSphere. This function provides improved load balancing in Parallel Sysplex DB2 data sharing configurations. With only the standard connection pooling included in WebSphere, an application must disconnect before another one can reuse a pooled connection. In a Connection Concentrator implementation, without any change in application behavior, the physical database connection is released at the end of a transaction (commit or rollback), instead of the closeConnection() call. This allows increased connection sharing and higher connection utilization because connections are not held by applications while not actively participating in a DB2 unit of work. The JDBC T4 sysplex workload balancing feature also implements a sophisticated scheduling algorithm that uses z/OS WLM information (passed back by DB2 UDB for z/OS) to distribute workload across members of the DB2 data sharing group at the transaction boundary.

In addition, connection concentrator will detect failed database connections and allocate a valid database connection when the application begins a new unit of work. This availability and workload management capability enables JDBC T4 to transparently relocate work away from failed or overloaded members to those that are up and underused, and to do so on a transaction-basis, rather than on a connection-basis. Also, unlike ordinary connection pooling, when an outage occurs on one of the members in the DB2 data sharing group, only those client connections that were actually processing transactions in that failing member receive connection failures. With ordinary connection pooling, all client connections to that member would receive connection failures regardless of whether the clients were active in the database server. Note that JDBC T4 sysplex and connection concentrator support is distinct from WebSphere Application Server connection pooling. Both can be used simultaneously as they don't conflict with one another.

Dynamic VIPA (DVIPA): Each link to an IP network must have an IP address. However, if a server's physical adapter or the link associated with a source or destination IP address fails, all TCP/IP relationships are broken until the adapter (and its IP address) is restored to service.

Virtual IP address, or VIPA, provides an IP address that is owned by a TCP/IP stack, but not associated with any particular physical adapter. Because the VIPA is associated with a virtual device, it is always available as long as the TCP/IP stack is functioning and accessible. This is useful for large systems such as System z machines that have multiple IP adapters, including OSA-Express and Hipersockets. As long as one IP adapter is working and connected to the IP network, others can fail without disrupting service to the DB2 data-sharing group.

But what happens if the TCP/IP stack itself becomes unavailable (for example, if a system loses power)? In this case, the VIPA goes away with the TCP/IP stack and is unavailable until the TCP/IP stack can be restarted.

Unlike a static VIPA, a DVIPA can move from one TCP/IP stack in the sysplex to another, automatically. TCP/IP stacks exchange information about DVIPAs, their existence, and their current location, and the stacks are continuously aware of whether their partner sysplex stacks are still alive and functioning so that they can back each other up. There are two different types of DVIPAs: multiple instance and application-specific. DB2 exploits both types: multiple instance DVIPA for DB2 group-wide new workload requests and the application-specific DVIPA for member-specific requests. The member-specific DVIPA is used by JDBC T4 to route workload to a specific member and a second member-specific port is used to allow two-phase commit failure and recovery processing.

In a member-specific DVIPA, if a DB2 data sharing group member is started on any system in the sysplex, the TCP/IP stack on that system will detect DB2 trying to bind a socket to the DVIPA. The properly configured z/OS TCP/IP stack (through the BIND parameter on the PORT statement) is smart. It will verify that the DVIPA is not active elsewhere in the sysplex, then will activate the DVIPA automatically before reporting successful completion back to DB2. If the DB2 member later fails, TCP/IP will deactivate the DVIPA. If that DB2 member is restarted on another TCP/IP stack in the sysplex that has been similarly configured to allow this dynamic activation, then the DVIPA will be activated there automatically so that the DB2 restart can proceed successfully. In other words, the member-specific DVIPA will follow its associated DB2 member around the sysplex. Each DB2 data-sharing group member will have a unique DVIPA.

Sysplex Distributor: Sysplex Distributor is the strategic IBM solution for IP connection workload balancing in the Parallel Sysplex. It is similar in concept to the IBM WebSphere Edge Components Load Balancer or the Cisco Multi-Node Load Balancer. It provides a cluster IP address for the entire DB2 UDB for z/OS data sharing group. Since it is built on Dynamic VIPAs, it is called a Distributed DVIPA. DB2 refers to this as the *location* or *group* Dynamic VIPA. When coupled with JDBC T4 and its sysplex workload balancing support, the Sysplex distributor IP address is only used for initial connection to DB2. All subsequent JDBC T4 requests use the member-specific IP addresses returned to the JDBC T4 client by DB2 UDB for z/OS.

The group DVIPA is defined on a primary TCP/IP stack (or "routing" stack) in the sysplex through coding of the VIPADEFINE and VIPADISTRIBUTE profile statements. Information about the DVIPA is distributed automatically to all designated candidate "target" TCP/IP stacks in the sysplex, and on all these target stacks, the same IP address is activated as a "hidden" DVIPA. The address is hidden in the sense that the target stacks don't advertise the presence of this IP address to the network. Only the routing stack advertises ownership of the group DVIPA to the world. Then, the routing stack waits to receive connection requests.

When DB2 data sharing group members start up, they bind their contact port (446) to the group DVIPA on their local (target) stack. When this happens, the target stack notifies the routing stack. The routing stack then knows it can send future DB2 connection requests to that target stack. If the DB2 member should fail, its target stack is immediately aware of this, and notifies the routing stack. These updates to the target stack are virtually instantaneous (sub-second), and do not rely on any sort of advisor function to make periodic queries for availability status.

When the group DVIPA routing stack receives a TCP connection request, it does the following:
1. Consults z/OS Work Load Manager (WLM) to find the relative available capacities on the z/OS nodes hosting the target stacks and DB2 members.
2. Consults the z/OS Communication Server Service Policy Agent for network performance and defined policies that might affect the distribution decision.
3. Selects a target stack and forwards the request for processing.

Should a target TCP/IP stack fail, in-flight connections also fail. The routing stack immediately sends out a connection reset to avoid the delay associated with TCP connection timeouts, and reconnect requests are sent to one of the surviving target stacks. New connection requests are not routed to the failed target stack or system.

Backup routing stacks may also be configured by coding the sypslex cluster IP address on a VIPABACKUP profile statement on the backup TCP/IP stack(s). If the routing stack suffers an outage, the backup routing stack takes over global responsibility for the group DVIPA. Each target stack is aware of the takeover, and sends the new routing stack its current connection routing table. The new routing stack consolidates all of the individual tables into its overall connection routing table. Any in-flight inbound TCP data lost with the failing routing stack is retransmitted by the individual client TCP stacks, and aside from a momentary pause for retransmission, clients and surviving servers continue without connection outage. This is all handled by collaborating Sysplex Distributor TCP/IP stacks; DB2 is entirely unaware of the failure of the former routing stack. When the original routing stack is restarted, it non-disruptively takes over its duties from the backup stack.

WLM: z/OS WLM allows you to define performance goals and assign a business importance to each goal. You define the goals for work in business terms, and WLM decides how much resource, such as CPU and memory, should be given to the application to meet the goal. WLM will constantly monitor the system and adapt processing to meet the goals. It can not only help with directing new work to available resources within a Parallel Sysplex, but it also directs resources to where they are needed to process existing work. When workload arrives at the JDBC T4 client, a workload classification occurs which can be based upon a number of factors including authorization ID, accounting strings, client information, stored procedure invocation, and so on.

ARM: Restarting a DB2 UDB for z/OS data sharing group member after it fails is important, as this will help free up any locks it holds as soon as possible and thereby minimize impact of the failure on other members of the DB2 data sharing group. The z/OS Automatic Restart Manager (ARM) helps ensure this restart process is carried out quickly and efficiently, based on a user-defined policy.

SFM: z/OS Sysplex Failure Management (SFM) allows you to optionally define a sysplex-wide policy that specifies actions z/OS is to take when certain failures occur in the sysplex. The goals of failure management in a sysplex are to minimize the impact that a failing system might have on the sysplex workload, so that work can continue with little or no operator intervention. One of several functions it can perform is to detect and isolate a system that has failed or is no longer responsive (that is, a “hung” system), remove it cleanly from the sysplex, and free up resources while ensuring that data integrity in the sysplex is preserved. This can be done either entirely automatically without any human intervention, or by prompting an operator for their go-ahead before proceeding.

Flow: The DRDA (Distributed Relational Database Architecture) protocol used by JDBC T4 provides an initial contact port and a resync port. For this architecture, JDBC T4’s initial contact port (DB2 port 446) is bound to the group DVIPA. To address the case where a DB2 member terminates and is restarted on another image (possibly alongside another DB2 member), each DB2 member configures its restart address with a resync port number unique to that member. Each member’s resync port needs to be reachable through a specific IP address to ensure correct resynchronization after a failure, so it is bound to a member-specific DVIPA.

After an initial connection is established, JDBC T4 can perform its own load balancing for subsequent work among the available DB2 data sharing group members. This is accomplished by the DB2 member sending back a list of IP addresses, one for each DB2 member in the data sharing group. Because the initial contact listening socket is listening on a group DVIPA, each DB2 member opens an additional listening socket on the same port as the group listening socket (446), but is bound to its member-specific DVIPA. It’s these DVIPA addresses that are sent back to the JDBC T4 client. JDBC T4 then balances across these member-specific DVIPAs, based upon information provided to it through WLM.

To summarize, the connection flow goes as follows:

1. JDBC T4 running within a WebSphere Application Server contacts the group DVIPA to make an initial connection. This connection request may travel over any of several physical network adapters owned by the TCP/IP stack to which the Distributed DVIPA is associated.

2. Sysplex Distributor fields the request. It consults with WLM and Service Policy Agent to decide which DB2 data sharing group member to send that connection request to, then sends it to the group member that has the most available capacity (per WLM goals, and so on.) to handle this first request.

3. The receiving DB2 member handles the request and responds back to JDBC T4 directly. Also included in its response is a list of member-specific DVIPAs, one for each DB2 member in the sysplex, and WLM recommendations on where to send the next request.
   After this point, sysplex distributor is no longer involved.

4. JDBC T4 performs its own load balancing across the member-specific DVIPAs, based on WLM data it receives from the DB2 data sharing group members. It frequently receives updates on the status of members of the DB2 data sharing group that it can use to make load balancing decisions. Because connection concentrator is being used, each routing decision is made on a transaction basis, rather than a connection basis. Because DVIPAs are used, connections to any given DB2 member can travel on any of several network adapters owned by the TCP/IP stack with which the DVIPA is associated.

5. If a DB2 data sharing group member (or its host TCP/IP stack or z/OS) fails, WLM will be aware of it, so JDBC T4 will be notified and will route new transactions to surviving DB2 members who all share concurrent read/write access to the same data as the failed member (by virtue of their

participation in the Parallel Sysplex data sharing group).

Based on user-defined policy, z/OS ARM may restart the failed DB2 member in place or on another z/OS system in the sysplex. Because each DB2 member has its own unique resync port, the DB2 member can even be restarted on a z/OS image in which another DB2 member is already active. When the DB2 member restarts, it will free up any held locks on shared data, its member-specific DVIPA will be re-enabled, and it will become accessible as before.

If an entire z/OS system failed or is not responding, then based on user-defined policy, z/OS SFM can also perform system isolation to cleanly remove the system from the sysplex.

6. If a network adapter fails, surviving adapters will automatically be used to access the affected member-specific or group DVIPAs.

7. If the routing TCP/IP stack that owns the group DVIPA fails, another stack in the sysplex will take over ownership for the group DVIPA and become the new routing stack.
   Any in-flight inbound TCP data lost with the failing routing stack will be retransmitted by the individual client TCP stacks, and aside from a momentary pause for retransmission, clients and surviving servers continue without connection outage.

Product Versions

This architecture requires the following software versions:

- JDBC type 4 driver level 2.7 or above is required to get the sysplex workload balancing and connection concentrator functions. This ships with DB2 V8.2 FP3, a.k.a. DB2 V8.1 FP10 or above.
- DB2 Distributed Data Facility for z/OS V6 and V7 support for DVIPA and Dynamic DVIPA is provided through APAR PQ46659.
- Base Dynamic VIPA support is available with OS/390 V2R8 and higher.
- Base Sysplex Distributor support is available with OS/390 V2R10 and higher. Support for integration of Sysplex Distributor with Cisco MNLB function (not shown in this architecture) is available with z/OS V1R2 and higher. Fast connection reset support is available with z/OS V1R2 and higher.

Planned Outages

This section discusses how each of the components can be taken down for software upgrades or any other planned outage.

Planned Outages	
Component	Procedure
Load Balancer	Same as in previous architecture.
HTTP Server	Same as in previous architecture.
WebSphere Application Server	Same as in previous architecture.
DB2 z/OS	1. Stop the DB2 z/OS data sharing group member you wish to upgrade or service. It will quiesce current work and notify the remaining DB2 group members and WLM that it is no longer available to process new requests. JDBC Type 4 driver and Sysplex Distributor will be told to route new requests to the surviving DB2 group members 2. Perform whatever maintenance is desired on the DB2 member. 3. Restart the DB2 member. Sysplex distributor and the JDBC Type 4

	driver will be notified that the DB2 member is available and will begin routing work to it. 4. Repeat with remaining DB2 z/OS data sharing group members, one at a time.
Sysplex Distributor	When the TCP/IP routing stack is brought down, its backup will takeover. The takeover occurs non-disruptively, and when the stack is restarted it automatically reclaims status as the routing stack.

What we Learned in Testing

When this architecture was set up and tested for planned and unplanned outages we learned the following:

- Did the software failover as expected? Yes.
- Did users experience any outage time or transactions that they needed to retry? No
- Did users experience any permanent data loss? No
- How long did the failover take? (How long did users experience outages): Instantaneous. Users do not perceive an outage.
  Details:
  - Stopping one of the members of the datasharing group, did not impact the performance of the workload coming to WebSphere. The workload continued to run at the same pace without any transactions ending in error or timing out. We see the following message in the WebSphere SystemOut.log:
    ```
    A connection failed but has been re-established. The hostname or IP
    address is "J90VIPA.pdl.pok.ibm.com" and the service name or port
    number is 446 . Special registers may or may not be re-attempted.
    ```
    This indicates that WebSphere recognized the failed connection and was able to re-establish a connection to a surviving member of the datasharing group, through the JDBC type 4 driver.
  - After re-starting the failed datasharing group member, it started accepting work again, without any interruption to WebSphere.
  - Refer to the "*System z Platform Test Report for z/OS and Linux Virtual Servers*" for details on how to setup the JDBC type 4 driver for Sysplex awareness.
  - There are some configuration errors that can make the Sysplex datasharing failover not work, or perform poorly. Please refer to the "*System z Platform Test Report for z/OS and Linux Virtual Servers*" for details.

Architectural Decisions

DB2 Connect EE vs. JDBC Type 4. Rather than using the JDBC Type 4 driver to contact DB2 UDB for z/OS in the Parallel Sypslex directly, each WebSphere server could be configured to have its JDBC driver (type 2 or type 4) go through DB2 Connect EE. A common, intermediate DB2 Connect EE server running in its own Linux guest image under z/VM could be configured. DB2 Connect would be used to communicate with the DB2 UDB for z/OS data sharing group members, utilizing capabilities similar to JDBC T4 for the sysplex workload balancing and the connection concentrator functions.

A backup could be configured for the DB2 Connect EE server. Failover could be configured for the DB2 Connect server, such that if the TCP/IP connection to the DB2 Connect server is lost, the client would automatically attempt to reestablish the connection. The client would first attempt to reestablish the connection to the original server. If the connection is not reestablished, the client would fail-over to an alternate DB2 Connect server (note that this client-level reroute to alternate DB2 Connect servers is not supported for two-phase commit (XA) workload from WebSphere. XA workload requires that the failed DB2 Connect server be restarted because transaction status information is stored within the DB2 Connect SPM log).

However, with the recent introduction of the sysplex workload balancing and connection concentrator functions in JDBC T4, it becomes the preferred means for driving traffic from WebSphere on Linux to DB2 in a Parallel Sysplex. Removing DB2 Connect from the picture in favor of JDBC T4 eliminates another potential point of failure, reduces administrative burden, and will improve performance for average workloads. Note that while JDBC T4 removes the need for DB2 Connect to be installed, a DB2 Connect EE license is still required in order to enable JDBC T4 to work directly with DB2 UDB for z/OS.

DNS/WLM vs. Sysplex Distributor. DNS/WLM solves the same problem as Sysplex Distributor – distributing workload across the sysplex by distributing client connections among active server instances. With the DNS/WLM approach, intelligent sysplex distribution of connections is provided through cooperation between WLM and DNS (Domain Name Service). For customers who elect to place a name server in a z/OS sysplex, the name server can use WLM to determine the best system to service a given client request.

In general, DNS/WLM relies on the hostname to IP address resolution for the mechanism by which to distribute load among target servers. As a result, the single system image provided by DNS/WLM is that of a specific hostname. Note that the system most suitable to receive an incoming client connection is determined only at connection setup time. After the connection is made, the system being used cannot be changed without restarting the connection. This solution is only available with the BIND 4.9.3 name server and not with the BIND 9 name server. DNS/WLM works for both TCP and UDP; Sysplex Distributor works only for TCP. Note also that many clients and intermediate DNSs cache IP addresses, so the client might continue to try to reconnect to an IP address that is unavailable until the owning and supporting TCP/IP stack is restarted, thus interfering with the availability provided by this approach.

Sysplex Distributor is IBM's strategic solution for connection workload balancing in a sysplex, so it was chosen over the DNS/WLM approach for this architecture.

Inclusion of Sysplex Distributor: In this architecture, Sysplex Distributor is only used for the initial contact between DB2 Connect EE and DB2 z/OS. After that contact, it steps out of the picture and allows those two entities to communicate directly. You can choose to not implement Sysplex Distributor. In this case, the choice of the initial DB2 z/OS DVIPA to be contacted could be predefined to JDBC T4. This approach would work, but requires that initial DB2 z/OS to be active in the Parallel Sysplex during initialization of the workload. Alternatively, some other workload routing technology (such as Cisco's Multinode Balancer) could be used instead of Sysplex Distributor to make the initial routing decision.

DB2 Connection Pooling vs. DB2 Connect Connection Concentrator: You could choose to use ordinary DB2 connection pooling rather than connection concentrator. But connection pooling has a drawback when used in conjunction with Parallel Sysplex. With Parallel Sysplex awareness activated in JDBC T4, JDBC T4 processes information about the availability of the members in a DB2 data sharing group only on the creation of a new connection to DB2 z/OS. With connection pooling also activated, there is a chance that connections would remain with a particular member of the data sharing group even if that member had problems. JDBC T4 would also not use WLM information to determine which of its pooled connections would be the best connection to be reused for a new request.

Connection concentrator builds on the features of connection pooling, preserving and enhancing connection pooling's performance benefits while eliminating the above weaknesses. Therefore, it was chosen for this architecture. Note however that applications that do not release resources, such as CURSOR WITH HOLD, TEMP TABLES, or bound with KEEPDYNAMIC(YES), do not release the physical connection.

Key restrictions or problem areas

There is no means of telling JDBC T4 to give preference for routing work to DB2 members located on the same physical System z server that are reachable over high speed hipersocket connections. JDBC T4's workload balancing decision is focused on the ability of the DB2 data sharing group members on z/OS to process the work, with no consideration being given to how long it will take to transport the request/response data between Linux and z/OS. As a result, work may be frequently sent to a DB2 data sharing group member on a different physical System z server, in which case an OSA connection, rather than a lower-latency Hipersockets connection, will be used for communication.

Solution estimating guidelines

Use the "WebSphere on Linux on System z Sizing" process to estimate the number of IFLs required for HTTP and WebSphere. Use the "Size390" process for sizing the number of standard CPUs required for DB2 z/OS.

Chapter 7: Reference Architecture: WebSphere with DB2 database on z/OS, in separate cities. (GDPS® /PPRC Multiplatform Resiliency for System z with HyperSwap™ Manager)

Scenario Being Solved

You have a key Web-connected, WebSphere application that runs in Linux on System z. The primary database for this application is DB2 for z/OS deployed in a data sharing group running in a Parallel Sysplex on IBM System z9 or System z CPUs.

The deployment aims not only to provide near-continuous availability by protecting from the failure of any one component, it also aims to protect, and to recover from, a Site 1 failure in which the applications and the production data are lost. In this architecture the appliances, the WAS application, zVM, DB2 and z/OS components are spread across two active sites that share the processing of the application's work. System A is in site 1. System B is in site 2. The primary production data runs in Site 1 and is mirrored to the secondary disks in site 2. If there is a Site 1 failure, processing will continue at Site 2 using the mirrored copy of the data. After the disaster, after Site 1 is restored, automation will restart the software, resync the primary and secondary databases, and switch network and data access to the original configuration in Site 1. Automation will also restore Site 2 to the original configuration.

From a software perspective, the front end, WAS, Linux for System z and zVM software deployments are described in Chapter 4 and the DB2 deployment is described in Chapter 6. Added to the picture are the components of the GDPS® /PPRC Multiplatform Resiliency for System z with Hyper Swap™ offering that provide the monitoring, operating system and application restarts, and the data mirroring and switching technologies required in the event of a site failure or planned downtime and recovery.

The key elements that this architecture adds to the architecture in Chapter 6 are the following:

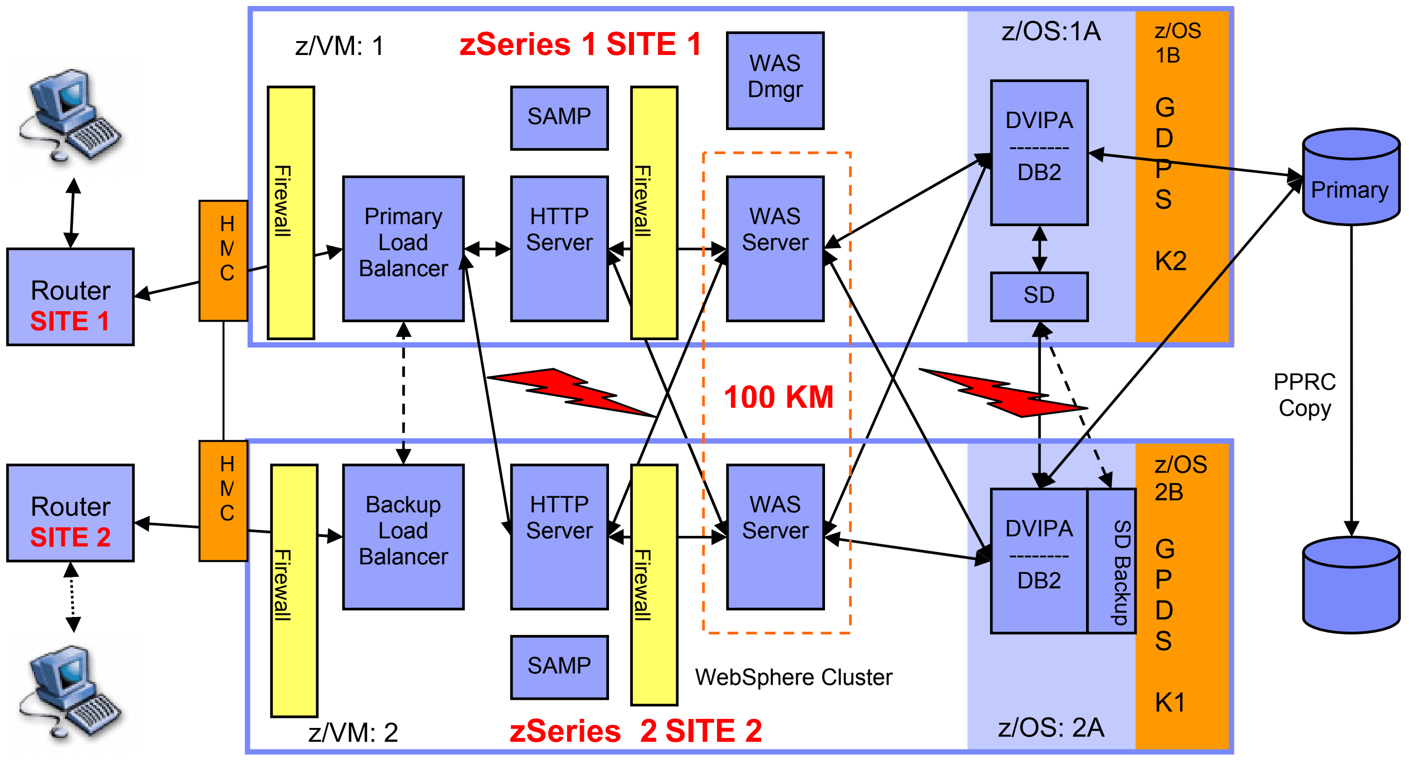
- Two data centers that are:
  - far enough apart to not be influenced by a single disaster
  - within 100 km to allow for synchronous data replication by way of two diverse fiber routes
- Production workloads configured to run at both sites with no primary or backup site.
- Primary database storage ESS/DS replicated to the second site using synchronous PPRC, maintaining data consistency between the primary and secondary devices on the two sites.
- HyperSwap function the in operating systems that enables the surviving z/OS and ZVM systems to switch to the secondary disks. The switchover is transparent so that the applications do not need to be quiesced. Note that for z/OS the switchover comes from a dynamic re-direction of I/O (UCB) pointers. For z/VM the HyperSwap function allows the dynamic swapping of virtual devices associated with one real disk to another real disk.
- Network connectivity and routing capability at both sites.
- GDPS 3.1 Control System software running at both sites to monitor systems and control failover and fallback of the network, systems and data.
- IBM Tivoli System Automation Mulitplatform (SA MP) running in Linux for zSeries and on z/VM. This monitors the functioning of the Linux, front end and WAS applications and can restart them, either in place or on another system in the same or different site.
- Required IGS Services (These ensure that the original setup is efficiently and accurately done and that the client staff is properly trained to become self-sufficient. It also leads to long term costs efficiencies, including the fact that GDPS warranty costs are less than standard.)

Architecture Principles

This architecture is designed to follow these principles:

- Any failures in the servers that make up this architecture will not be noticeable to the user. The current transaction may fail, but subsequent transactions succeed. After any single failure, transactions may continue at the same rate with only a brief, or possibly no, degradation in throughput or response time.
- The architecture must be rapidly scalable to support increases and decreases in business volume.
- No single point of failure
  - Hardware: There should be no component within a site that would result in loss of end user function if it failed. There are exceptions, however. Where components are “hardened” from an availability point of view (usually by having internal redundancy like dual power supplies and so on), then a single component is allowed.
  - Facilities: There should be two geographically distant sites.
  - Software: There should be fallback application server(s) with enough capacity to carry on the essential work in the event of a site failure. This requires geographically dispersed operating systems, appliances, applications, and database environments.
- Data Management with data integrity
  - Data mirroring between sites is required.
  - Data mirroring and site switchover should both be performed at the lowest level (hardware) for speed, accuracy, and independence of any application.
  - Data consistency is required for fast workload restart.
  - Data mirroring freeze is required to prevent rolling database corruption.
- Automation
  - End-to end automation is required. Humans cannot be guaranteed to provide adequate monitoring and response, especially in times of disaster.

Reference Architecture



GDPS® /PPRC Multiplatform Resiliency for System z with HyperSwap™ Set up

The normal running configuration of GDPS /PPRC Multiplatform Resiliency for System z with HyperSwap when there is no disaster is:

- A GDPS/PPRC V3 system is running GDPS automation based upon Tivoli NetView and System Automation or z/OS at both sites. The primary Controlling System (K1) is running in Site 2:
    - Monitors the Parallel Sysplex cluster, coupling facilities, and storage subsystems and maintains GDPS status
    - Invokes HyperSwap
    - Invokes network switching, based on user-defined automation scripts
- System Automation Multiplatform (SA MP) proxy servers run in each VM partition and SA MP agents run in each Linux software component (firewall, load balancer, HTTP server, WAS Server). These provide GDPS automation with awareness of the state of the software running in the Linux guests and work with GDPS to restart them during the recovery phase.
- All of the operating systems must run on System z servers that are connected to the same Hardware Management Console (HMC) Local Area Network (LAN) as the Parallel Sysplex cluster images.
- All critical data resides on storage subsystem(s) in site 1 (the primary copy of data) and is mirrored to site 2 (the secondary copy of data) through PPRC synchronous remote copy.
- Site 1 system has connectivity to secondary disks
- Parallel Sysplex with DB2 production data-sharing systems is running in both sites 1 and 2.

Flow of requests through this architecture

Site1 Failure

If there is a failure at Site 1, the GDPS K1 System in site2 (using Sysplex communications) detects this and automates the failover process:

1. Invoke HyperSwap to switch the surviving z/OS and z/VM systems at the secondary site to use the secondary disks which contain the mirrored data.
2. Switch network connectivity to the Site 2 router, based on customer-provided scripts.
3. Take steps to ensure that 100% of the incoming workload can be handled by Site 2. Use one or all of the following options:
   - Stop expendable work in Site 2 LPARS to accommodate the primary site workloads.

   Or
   - Invoke On/OFF Capacity Backup (O/O CBU) to increase the number of IFLs available to the VM LPAR and the number of CPUs available to the z/OS partition.

   Note, as discussed in the section, Alternatives Considered, below, IFLs, CPUs and memory cannot be shared between the servers, so if it is required that failover take place immediately and without any interruption to the users, then each server at both sites must have sufficient IFLs, CPUs and memory to run 100% of the workload should the other server fail. This means that each server would run its production workload at less than 50% utilization. Software costs increase because of this.

   The software costs can be reduced by using O/O CBU to duplicate CPUs and IFLs online at the secondary site, should the primary site fail. However, it means that for the first few minutes after a failure, only 50% of the needed IFLs and CPUs are available to handle 100% of the workload. It will take approximately 5 minutes to bring the new IFLs online to z/VM. This solution gives lower cost as long as reduced performance can be tolerated until the capacity upgrade CPUs are brought online.

4. DB2 non-disruptively continues processing on z/OS 2A in Site 2.

Site 1 Recovery: *After* the site and the *System z* CPUs are restored*:*

1. Site 1 systems (z/OS, z/VM, Linux, WAS, and WAS applications) are restarted as needed.
2. DB2 synchronizes data between Site 2 and Site 1 disks.

   Note that the time for this can take anywhere from hours to days, depending upon the amount of data changed. GPDS/PPRC with HyperSwap v3.2 adds exploitation for Metro Mirror Failover/Fallback which essentially tracks all changes made to the data on the secondary disks so that when falling back only the changed data needs to be copied. This greatly reduces the fallback time.

3. GDPS restart of z/OS and workloads (DB2, Sysplex Distributor)), and zVM and Linux Guests. Stop unneeded systems in Site 2.
4. SA MP restart of Linux Guest workloads. Stop unneeded workloads in Site 2.
5. HyperSwap z/OS and z/VM (and Linux) switch connectivity to primary disks.
6. Switch network connectivity, if needed, based on customer-provided scripts.
7. Release On/Off Capacity Backup
8. DB2 non-disruptively continues processing using the restarted Site 1 resources.

Product Versions

This architecture requires the following software versions:

- z/OS currently supported releases
- DB2 currently supported releases
- GDPS V3.2 or higher, or V3.1 with enabling APAR for GDPS/PPRC Multiplatform Resiliency for System z
- z/VM v5.1 with the HyperSwap function
- Tivoli NetView for z/OS V5.5 or higher
- Tivoli System Automation for z/OS V2.2 or higher
- Tivoli System Automation for Multiplatforms R2
    - Fixpack (1.2.0-ITSAMP-FP02) for IBM Tivoli System Automation for Multiplatforms
- Linux For System z
    - SuSE SLES 8 refresh (4/2004 or newer)

Planned Outages

This section discusses how each of the components can be taken down for software upgrades or any other planned outage.

Planned Outages	
Component	Procedure
Load Balancer	Same as in previous architecture.
HTTP Server	Same as in previous architecture.
WebSphere Application Server	Same as in previous architecture.
DB2 z/OS	Same as in previous architecture
Sysplex Distributor	Same as in previous architecture
GDPS Control System	Invoked by operator commands.
Disk Subsystems	The planned HyperSwap Function provides the ability to transparently switch all primary disk subsystems with the secondary subsystems for planned reconfigurations. This enables planned site maintenance without requiring any applications to be quiesced.

What we Learned in Testing

The GDPS/PPRC Multiplatform Resiliency for System z with HyperSwap Manager has been tested at the Linux and z/OS operating system level and the failover and fallback is shown to perform as described. The WAS and other linux-based applications described in the preceding scenarios have not been specifically tested with GDPS/PPRC Multiplatform Resiliency for z with HyperSwap Manager, but their failover and fallback should not be impacted by this automated, cross site scenario. HyperSwap from primary to secondary disks occurs at the VM operating system level is transparent to the applications.

IBM recommends that the failover and fallback of all applications deployed in a GDPS/PPRC HyperSwap Manager environment be verified by testing on a regular basis.

Architectural Decisions

The key Architectural components selected to achieve the goal of cross-site disaster recovery are the GDPS control system, PPRC (Metro Mirror), and HyperSwap.

GDPS Control System

Geographically Dispersed Parallel Sysplex (GDPS) is an automated operations solution that runs in z/OS (the LPAR is often referred to as the K system, or Control System.) It must maintain its viability and thus, IBM recommends that it always run in its own z/OS LPAR in the Parallel Sysplex.

Using Tivoli System Automation for z/OS and its own local disks, GDPS can monitor planned and unplanned exception conditions and manage a controlled site switch for outages of z/OS and Parallel Sysplex, z/VM®, Linux for System z (as a guest under zVM) , and VSE/ESA™ systems and the applications that run in them.

In this scenario, GDPS/PPRC with Muliplatform Resiliency for System z will manage the HyperSwap to the secondary disks in Site 2, the optional activation of resources in Site 2 to provide equivalent capability as the lost Site 1, and the recovery of work in to Site 1, after the facilities and CPU systems have been restored.

GDPS includes standard actions to:

- Stop: quiesce a system's workload and remove the system from the Parallel Sysplex cluster (stop the system prior to a change window)
- Start : IPL a system (start the system after a change window)
- Recycle: quiesce a system's workload, remove the system from the Parallel Sysplex cluster, and re-IPL the system (recycle a system to pick up software maintenance)
- Manage Parallel Sysplex Coupling Facility Structures

The standard actions can be initiated against a single system or a group of systems. Additionally, user-defined actions are supported (for example, planned site switch in which the workload is switched from processors in site 1 to processors in site 2).

GDPS unplanned reconfiguration support not only automates procedures to handle site failures, but will also minimize the impact and potentially mask a z/OS system or processor failure based upon GDPS policy:

- If a z/OS image fails, the failed system will automatically be removed from the Parallel Sysplex cluster and the workload restarted on the second system in the sysplex. (See the illustration above. If z/OS: 1A fails, its work can be restarted on Z/OS: 2A.)
- If the hardware fails (zSeries 1 SITE1, for example), the z/OS image will be started up (IPLed) (on zSeries 2 SITE2) at the second site, and its workload restarted.

PPRC (also known as Metro Mirror)

Peer-to-Peer Remote Copy (PPRC) manages the ongoing synchronous remote data copy, freeze, and Flashcopy functions that make it possible to switch sites with no data loss, and full data integrity across multiple volumes and storage subsystems.

HyperSwap

HyperSwap™ function provides for a site switch of disk subsystems that eliminates the need for an IPL at the recovery site. In this architecture it substitutes PPRC secondary for the primary device and is controlled by GDPS automation. The GDPS/PPRC HyperSwap architecture can manage very small to very large DASD environments. For example, the HyperSwap component can manage an unplanned disk reconfiguration for 6,545 volume pairs (19.6 terabytes) in 15 seconds.

Note that Microcode-related disk failures are among the most common causes for system outages. GDPS/PPRC HyperSwap addresses this in a scenario where all of the production is in Site 1 with disk mirroring to Site 2. Any disk failure in Site 1 will trigger a swap to a secondary disk in Site 2 and the Site 1 systems will continue to run. If all of Site 1 is gone, PPRC freeze will be invoked and the environment can continue to run in Site 2.

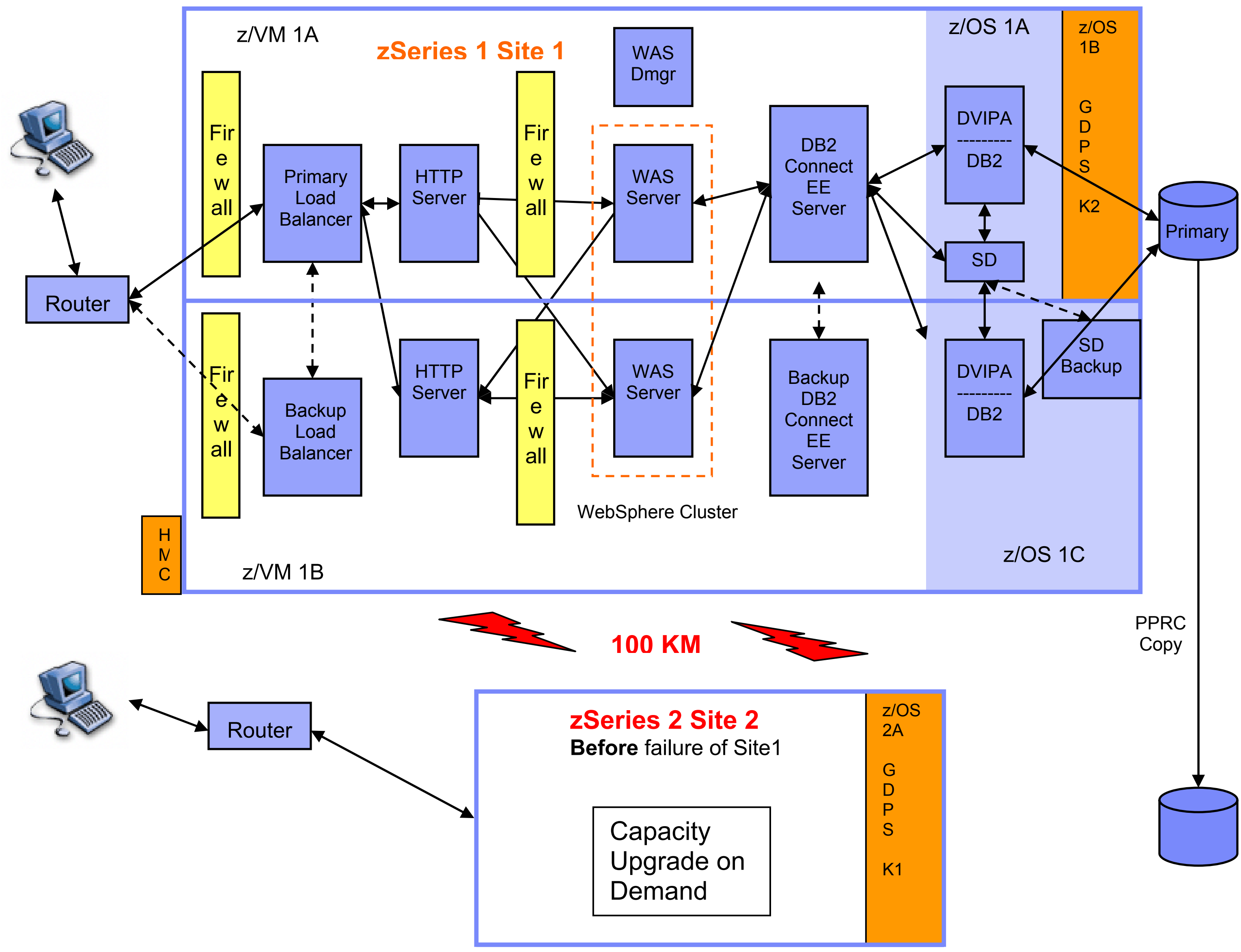
Alternatives Considered: Active - Passive deployment

Because IFLS and memory cannot be shared across System z systems and sites, each server must have sufficient IFLs and memory to run the workload if the other server fails. This can be costly to configure.

A potentially less expensive architecture that provides *near continuous availability* is depicted in the following diagram.

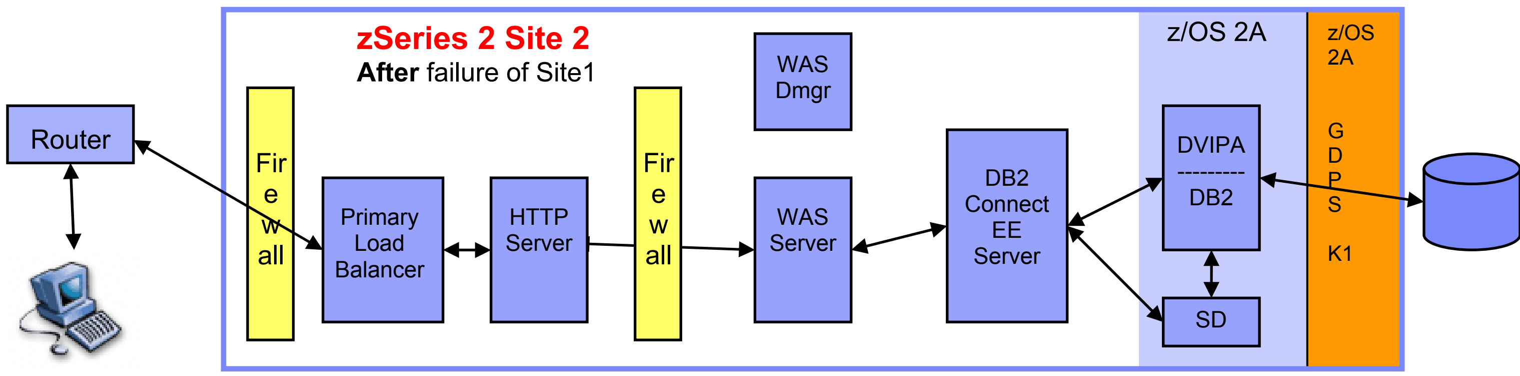
It assumes an “active-passive” model with production application servers at the primary site (Site 1) and standby server capability in the parallel sysplex at the secondary site (Site 2). Starting the standby servers on Site 2 lengthens the recovery time by the amount of time required to start the standby operating systems and applications. While a standby configuration lengthens application server recovery time, this design can take advantage of the System z Capacity Upgrade on Demand feature (bringing additional CPUs online at the secondary site only if needed), saving hardware and associated software costs.

Note that both architectures require the same primary and secondary disk configurations at the two sites and employ PPRC and HyperSwap as described above. They both require a K1 system running at the second site.



Before failure of Site1

After failure of Site 1



Key restrictions or problem areas

- IGS Services for Installation are required to ensure the proper set up and education of customer personnel. GDPS cannot be purchased independently.
- Achieving 100 km distance between sites requires a Sysplex Timer extender to be positioned at a third site between the two. Otherwise the Sysplex Timer allows for a 40 km separation.
- Site 1 Recovery: DB2 Coupling Facility Data Considerations:

  In a GDPS/PPRC environment, when coupling facility structures, such as those for DB2 Group Buffer Pools, are configured for duplexing across two coupling facilities (usually to avoid SPoFs or to provide structure recovery) and a primary site failure occurs, GDPS cannot ensure that the contents of the coupling facility structure are time-consistent with the data on DASD.

  For site 1 application restart and recovery processing GDPS must discard all coupling facility structures at the secondary site during the process of restarting the workload. The result is a loss of changed data in coupling facility structures.

  You must execute potentially very lengthy and highly variable DB2 data recovery procedures to restore this lost data. The length of time depends mainly on the number of objects that must be recovered, and the amount of logs that must be processed. IBM testing and customer experiences indicate that the recovery time is anywhere from a minute or less (10-30 objects and small amount of log) to two hours (thousands of objects and many logs).

- Synchronous Data Mirroring Performance Overhead

  Some transaction processing systems may not tolerate the amount of transaction processing overhead caused by PPRC data mirroring. Tests show that, for the DS8000 storage systems, over fiber channels, response time is 0.4 microseconds with an additional 10 microseconds for each km distance that the disk is from the host server. For example, if the host system is 5 km from the disk, response time increases by 50 microseconds (0.050 milliseconds)

- This two site architecture can require data encryption not required when a solution is deployed on a single System z server.

Solution estimating guidelines

Use the "WebSphere on Linux on System z Sizing" process to estimate the number of IFLs required for HTTP and WebSphere. Use the "Size390" process for sizing the number of standard CPUs required for DB2 z/OS.

1. Client View: The time and resources required to implement this Continuous Availability Reference Architecture depend upon the client's starting point. Typically, clients spend from 6 to 12 months depending upon the end of the spectrum from which they are starting. The order of implantation typically follows:
    a. Provision a second site
    b. Implement PPRC
    c. Implement a multi-site sysplex
    d. Implement GDPS

Critical skills required are for Parallel Sysplex, automation and remote copy.

Appendix 1: References

- *IBM System z Platform Test Report for z/OS and Linux Virtual Servers, June 2006 Edition (Part 3. Linux Virtual Servers).* This report details the setup and testing of these reference architectures. *www-03.ibm.com/servers/eserver/zseries/zos/integtst/*
- IBM Whitepaper: *Automating DB2 HADR Failover on Linux using Tivoli System Automation.*
- IBM Redbook SG24-6392: *WebSphere Application Server V6 Scalability and Performance Handbook.*
- IBM Redbook SG24-6688: *WebSphere Application Server Network Deployment V6: High availability solutions.*
- IBM Press, ISBN 0-13-146862-6: *WebSphere Deployment and Advanced Configurations,* Chapters 21, 22. Barcia, Hines, Alcott, Botzum.
- IBM Redbook SG24-6093: *PeopleSoft V8 on zSeries Using Sysplex Data Sharing and Enterprise Storage Systems*, Chapters 3 and 4, IBM, March 2004
- IBM Whitepaper: *Leveraging z/OS TCP/IP Dynamic VIPAs and Sysplex Distributor for Higher Availability*, Jay Aiken, July 2002
- IBM whitepaper: *Improve Your Availability With Sysplex Failure Management,* Riaz Ahmad, www-1.ibm.com/servers/eserver/zseries/pso/image/**sfm**.pdf
- IBM Document Number SC31-8775-02: *z/OS V1R4 Communications Server: IP Configuration Guide*
- IBM document number SC09-4835-01: *DB2 Connect Users Guide V8.2*
- IBM Redbook SG24-6517: *Communications Server for z/OS V1R2 TCP/IP Implementation Guide, Volume 5: Availability, Scalability and Performance*, Rodriguez, Fascini, et al, October 2002
- IBM GDPS Website: www.ibm.com/servers/eserver/zseries/gdps.html
- IBM Redbook SG24-6344: *Linux for zSeries: Fibre Channel Protocol Implementation Guide*
- IBM Redbook SG24-6694: *Linux for IBM System z9 and zSeries*
- IBM Redpaper REDP-0205: *Getting Started with zSeries Fibre Channel Protocol*

Comments on this paper.

Please address comments to:
Steve Wehr, swehr@us.ibm.com.

Appendix 2: Legal Statement