Redbooks Paper

Erich Amrehn
Ronald Annuss
Bernhard Dierberger
Richard Plasun
U. Sager

Linux on IBM zSeries and S/390: High Availability for z/VM and Linux

Preface

This paper provides information to help readers plan for and install a high availability solution for Linux for zSeries running under z/VM. It is intended for customers, technical presale people, and technical managers, to help them discuss the high availability possibilities of a Linux for zSeries environment running under z/VM and to guide them in implementing a solution.

Objectives

The objectives of this paper are:

- To introduce basic high availability concepts applicable for Linux for zSeries under z/VM
- To show sample scenarios that are basic components of a complex solution

Beyond the scope of this paper are:

- Attempting to cover all aspects of the application high availability.
- Integration in systems management.
- A discussion of total network high availability. For the purpose of this paper we assume that the network infrastructure outside the zSeries server is highly available.

To find the sections of this paper that are pertinent to specific high availability scenarios, see Table 1 on page 2.

Table 1 High availability topics

Scenario	Service level	Sections
z/VM failure	Few minutes	“z/VM view of high availability” on page 10 “z/VM high availability” on page 17
Application takeover	Few minutes	“Practical Linux high availability” on page 31
Data availability	Few minutes	“General topics for Linux and z/VM” on page 21
Adapter failure	Continuous	“How to handle an Open System Adapter (OSA) failure” on page 21 “OSA adapter failure” on page 42
z/VM failure	Continuous	“High availability of z/VM with network dispatcher” on page 29 “Customer scenario” on page 54
Application failure	Continuous	“Load-balancing cluster” on page 5 “Customer scenario” on page 54
Data high availability	Continuous	“Customer scenario” on page 54

Introduction

In today’s highly competitive e-business world, outages can have a devastating impact on a business. Maintaining maximum system uptime—high availability—is therefore becoming increasingly critical to business success.

The introduction of Linux for z/Series has caused many customer to think about implementing business applications on the highly reliable and scalable zSeries hardware. Linux for zSeries inherits the hardware’s reliability, but software faults can still cause outages.

The z/VM virtualization technology allows consolidation of a significant number of Linux servers. These Linux servers run under z/VM as virtual machines. z/VM virtual machines, guests, or z/VM userids are all basically the same and mean a runtime environment for a zSeries operating system.

An outage of z/VM means that all consolidated servers are affected. Therefore, special consideration for a z/VM outage must be made. A high availability solution only for Linux is not sufficient.

No high availability product currently exists that covers both Linux and z/VM requirements. Only Linux high availability products are available today.

Thus a serious need exists to discuss and demonstrate how a high availability solution on Linux for zSeries running under z/VM can be implemented.

Definition of high availability

Before discussing the high availability possibilities in a Linux on z/VM environment, it is important to provide some introduction to the subject of high availability.

A highly available system is a system that is designed to eliminate or minimize the loss of service due to either planned or unplanned outages. High availability doesn’t equate to continuous availability (that is, a system with nonstop service). You can achieve nearly continuous availability with, for example, the Parallel Sysplex technology. The eServer zSeries operating system z/OS exploits this technology to achieve continuous availability. Unfortunately z/VM and Linux can’t participate in a Parallel Sysplex.

The effort needed to implement a highly available system depends strongly on the service level agreement. Before designing for high availability systems, a service level agreement appropriate for your business must be defined. This means that a clear understanding of the allowed downtime and the context of how the downtime is measured must be achieved.

For the purpose of this document the following definition will apply: Highly available systems may fail but they are designed to recover quickly. Typical failover time depends on the number of servers and the size of data.

Techniques and terminology of availability

The basic rule of implementing a highly available environment is to identify and remove single points of failure. A general rule is to use redundant components (hardware and software).

Several aspects of availability must be considered: hardware, process/application software, data, and networks all contribute to the overall reliability of the system. Some basic considerations in each of these areas are presented in the next few sections.

Hardware availability

A general rule for building highly available systems is to identify and avoid single points of failure not only for the software components, but also for the hardware. The availability features of all aspects of the hardware must be considered, for example:

- Power supply
- CPU
- Memory
- Network adapter
- I/O subsystem

IBM zSeries hardware is designed for continuous availability, so zSeries systems offer a set of reliability, availability, and serviceability features (RAS). The zSeries architecture has two instruction units to compare every operation in order to discover possible errors. The main storage and the L2 cache uses Error Checking and Correction (ECC) technology. In addition, memory sparing is implemented to replaces failed memory units. CPU sparing is also available. A detailed description of the zSeries hardware can be found in the redbook *IBM eServer zSeries 900 Technical Guide,* SG24-5975.

Despite the zSeries RAS features, planning a failover solution for the whole zSeries server in a disaster case, or for failing network adapters and disk subsystems, is necessary. More details can be found in the section “z/VM high availability” on page 17.

Process/application high availability

This means that processes and applications are available to serve requests. In a simple case we only have to restart the process, but in a more complex application we have to implement some steps so that a fast and secure restart of the application is possible. If the application is not designed for high availability, we are not able to build a highly available environment. For example, in the case of a crash, if an application locks files that weren't released, these locks will influence the takeover process or even make failover nearly impossible.

An often-used approach to achieve application availability is software clustering with a network dispatching component in front of the application. In a highly available environment, some components must be provided to check the health of the application. A monitoring tool must be adapted to the specific application. In a complex environment, such as one with Enterprise Resource Planning applications, databases, and some middleware components, it is no easy task to monitor the various processes and to develop takeover rules.

Besides the application, the operating system and the network connection also must be monitored; without an operating system, applications can't run. This means that integration into a systems management concept must be developed to monitor the operation system and to automate the systems management procedures to protect the system from outages. For a Linux on zSeries environment, often two operating systems (z/VM and Linux) are involved, and therefore special considerations are necessary.

Data high availability

Data high availability means that data survive a system failure and are available to the system that takes over the failed system. Data high availability may not be equivalent to transaction safety. "Transaction safety" means that the application, with or without additional middleware products, ensures the data integrity (for example, with two-phase commit) even if a failure occurs in the middle of a transaction. For the purposes of this discussion, data high availability means that the data is highly available once it is stored onto disk.

Data high availability can be achieved by sharing the data across the systems. In the zSeries environment you can configure different channels to I/O subsystems, thus the devices are available to different Linux guest systems.

The general data types require different actions to maintain data high availability. These types are:

- Static data binaries, install images, and so forth
- Active data, meaning data that is rapidly changing

Data availability in the simple static data case can be achieved with shared Direct Access Storage Devices (DASD) because only read requests are involved. This only helps if one node fails. If the disk itself is not available due to a control unit error or a disk failure, one needs a backup of the disk's subsystem.

One potential solution is to mirror the DASD to an second subsystem and switch to this subsystem. This requires a backup or mirroring procedure for the data. Depending on the size and the data type, several option exists.

One of these options is the open source Linux rsync tool, which is able to mirror data from one site to another. You can periodically run the tool to synchronize the data. The solution with rsync is only suitable with static data and relative small amount of data to synchronize.

For active data, a combination of the Linux network block device and software RAID can provide a online data mirroring solution. The network block device is a Linux device driver that provides access to physical disks across the network. With the Linux software RAID, multiple disks can be assembled into a disk array. For more information see the redbook *Linux for zSeries and S/390: ISP/ASP Solutions,* SG24-6299.

Hardware storage solutions exist for synchronous and asynchronous backup/mirroring support. The IBM Enterprise Storage Server (ESS), for example, provides remote copy services. These are:

- Peer-to-Peer Remote Copy (PPRC)
- Extended Remote Copy (XRC)
- FlashCopy

- Concurrent Copy

XRC and Concurrent Copy rely on the so-called System Data Mover (SDM) program, which is a component of z/OS.

The remote copy features of ESS allow disks to be geographically distributed, and offer highly available data storage. For more information see the redbook *IBM Storage Solutions for Disaster Recovery,* SG24-6547*.*

Network high availability

A very important issue is to build up a resilient network infrastructure. Implementing failover pairs to provide network adapter fault tolerance is a simple and effective approach to increase the reliability of server connections. Redundant router configurations, hardware- and software-based load balancing techniques with switches/routers, and so forth, support the implementation of a highly available network infrastructure. Moreover, the network cable could also be a single point of failure.

High-availability clusters

High-availability (HA) clusters exist to keep the overall services of the cluster available as much as possible, taking into account the failure of hardware and software. If the primary node in a high-availability cluster fails, it is replaced by a secondary node that has been waiting for that moment. That secondary node is usually a mirror image of the primary node, so that when it does replace the primary, it can completely take over its identity and thus keep the system environment consistent from the user's point of view.

Load-balancing cluster

Another cluster type is the load-balancing cluster. Its main purpose is to spread incoming traffic to more than one server. The focus of a load-balancing cluster is to achieve a scalable solution and to easily put new resources into the environment when the load exceeds the actual capacity. If a load-balancing cluster is also used for high availability, you have to plan for more hardware resources on the cluster nodes to provide the expected performance of the system.

In a load-balancing cluster, an application with a specific configuration runs on more than one node. If a node fails, there is nothing to failover since the applications are already active on the other nodes.

Linux kernel support for load-balancing techniques is provided from the Linux Virtual Server project (LVS). The Linux Virtual Server is built on a cluster of real servers, with the load balancer running on the Linux operating system. The architecture of the cluster is transparent to end users. End users only see a single virtual server. For more information, see the Linux virtual server Web site at:

`http://www.linuxvirtualserver.org`

More information on how to build a load-balancing cluster for high availability Linux on zSeries clusters, see the redbook *Linux for eServer zSeries and S/390: Distributions,* SG24-6264.

Cold standby

In the literature, several definitions of the term “cold standby” are published. For the purpose of this paper the following will apply: A high availability environment consists of a minimum of 2 servers. In a cold standby system the redundant component is in an empty state and must be initialized to bring it online. In our cold standby test scenario, the two Linux images are active only if the application is not started on one server.

Hot standby

For the purpose of this paper we define the term hot standby as follows: Hot standby is a scenario where the secondary components share some state with the active server. In case of a failure, the takeover time is reduced compared to the cold standby scenario because the second application is already started.

High availability example

Let's consider an example of a high availability two node cluster to illustrate the basic technologies and procedures to build up a high availability solution.

A typical high availability environment consists of:

- Two or more Linux for zSeries systems in one z/VM, or in two Logical Partitions (LPAR) on the same zSeries processor complex, or on different zSeries servers
- A set of shared disks for data availability, or a network file system
- At least two network connections
- A heartbeat and/or monitoring tool on each node

Figure 1 summarizes a typical HA system. Each node has two network interfaces (service and standby) and in addition, a non-IP heartbeat connection. The active node is called the primary or service node; the other is the secondary or backup node.

The non-IP connection is usually used to check the health of the nodes. It is also possible to use the normal network connection for the heartbeat, but an additional non-IP connection is preferred because it does not rely on the IP stack, so a failure of the network adapter can be identified. Since the system can differentiate between a node and a network failure, more sophisticated error handling routines are possible and the negotiation between the two nodes in case of a network failure can still happen. This is especially important for the proper release of shared resources (such as disks) before the takeover occurs.

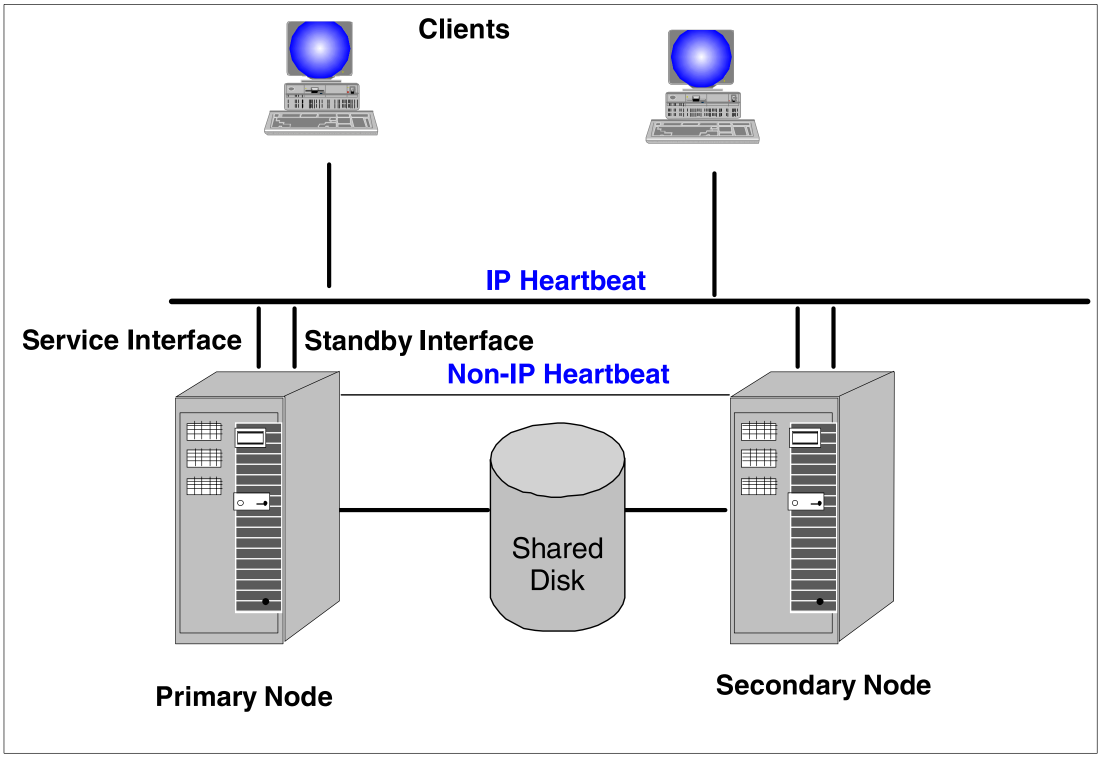

Figure 1 Two node high availability cluster

The business-critical data reside on a shared disk. The zSeries has no internal disk, thus an external shared disk subsystem is always available.

If the active node fails, the heartbeat tool detects the failure and performs the IP, data, and application takeover. Both servers have a real IP address for each adapter. The application connects to the server with a so-called virtual IP address. The virtual IP address is an alias address defined on top of the real network interface. Both real IP addresses are visible from outside and are incorporated in the address resolution protocol (ARP) to provide the IP-to-MAC mapping. During the IP takeover the virtual IP address is transferred to the second node and an ARP message (gratuitous ARP) is sent to inform the other computers in the network of the new active node. The heartbeat tool has to provide scripts to handle the necessary actions such as start/release and stop the resources.

The situation where the two nodes each falsely believe the other to be dead is called a partitioned cluster. If both have shared write access to the disks, data integrity can be damaged. One way to ensure data integrity is called STONITH[1]. STONITH achieves data integrity by stopping one node, and therefore only one node has access to the data.

Disaster recovery vs. high availability

Disaster recovery refers to how a system recovers from catastrophic site failures. Disaster recovery requires a replication of the entire site. The recovery time in case of a disaster is in the range of hours. The tasks necessary to recover from disaster differ from those needed to achieve a highly available system.

[1] Shoot the other node in the head.

Disaster recovery: Dual site concept

A common technique for disaster recovery is the “dual site concept,” where two data centers reside in different locations. The entire hardware configuration is redundant, and the two systems are connected to each other.

Figure 2 shows a typical dual site environment with duplicated hardware (CPUs, DASD, and tapes). In this example two Logical Partitions (LPARs) with z/VM and Linux guest systems are shown on each site. Data mirroring is necessary to accomplish data recovery in case of an I/O subsystem failure. To enable a site takeover to take place, the sites must be connected to each other with separate connections.

The IBM ESS has implemented the Point-to-Point Remote Copy (PPRC) feature of the zSeries disk controller hardware. PPRC allows a remote disk unit to hold a synchronized copy of the data stored on a production disk subsystem. The synchronization of the data takes place at the time of the I/O operation; thus the remote copy is identical to the production data. In case of a failure of the production disk subsystem, the backup disk subsystem can be brought online and the processing resumes at the point of failure. For a more detailed discussion on this topic see the redbook *IBM Total Storage Solutions for Disaster Recovery SG24-6547*.

PPRC can be used to allow a site takeover. The logical process to perform a takeover is as follows:

1. On the recovery site (secondary): Start the system image that will be used as the system during the outage.
2. On the primary site: Stop the applications and guests to be moved, to guarantee no I/O on PPRC volumes.
3. Delete the PPRC pairs to return them to simplex state.
4. Delete the PPRC paths.
5. Establish PPRC paths from the secondary ESS back to the primary.
6. Reverse the direction of the PPRC pairing.
7. Suspend all PPRC volumes. This breaks the pairings.
8. Shut down the original system.

Use the following steps to return to the primary location:

1. Copy all changed data back to the original primary ESS.
2. Stop applications and guests to guarantee no I/O on PPRC volumes.
3. Delete the PPRC pairs to return them to simplex state.
4. Delete the PPRC paths.
5. Establish PPRC paths from the primary ESS.
6. Establish pairing from the original volumes (reverse the direction of the PPRC pairing).

PPRC commands are not directly supported from the Linux shell, but you can implement ESS PPRC on your z/VM systems. The control and management functions of PPRC are done with the device support program ICKDSF of z/VM, or with the ESS Web interface, which can also be used in a Linux-only environment. However, ICKDSF contains only a subset of the commands and functions available in a z/OS environment. You must establish procedures to control recovery in the event of a disaster—for example, procedures to respond to failure scenarios.

Automated site takeover requires a thorough understanding of the whole environment, as well as the use of some advanced procedures. Implementing such procedures is no easy task, and is beyond the scope of this paper.

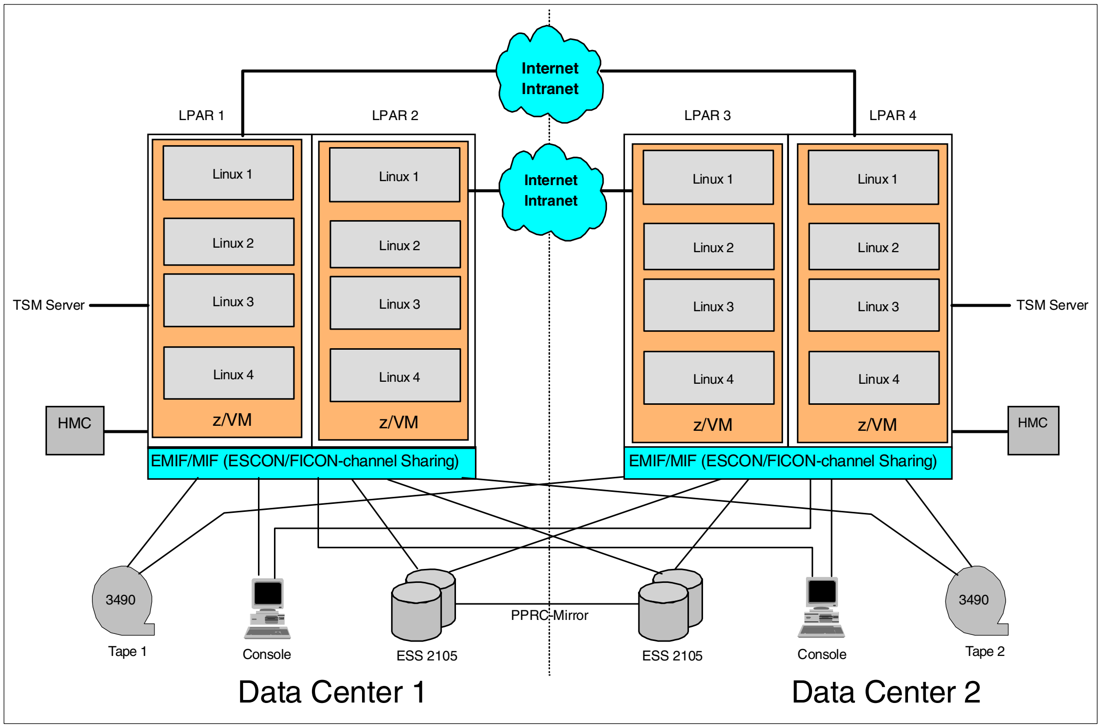

Figure 2 Disaster recovery dual site concept

z/VM view of high availability

Most Linux for zSeries installations use z/VM as the platform of choice since z/VM provides the necessary flexibility and robustness for running a lot of Linux images on a zSeries server. In this section we discuss some basic z/VM features and tools that are necessary or helpful for developing a high availability solution for Linux servers running on zSeries in a z/VM environment.

z/VM overview

z/VM is an operating system for IBM zSeries servers with a rich set of services in support of interactive users, client/server environments, and with the capability to run full-function operating systems such as z/OS, VSE, or Linux for zSeries.

The two most important components of z/VM are the control program (CP) and the conversational monitor system (CMS). CP virtualizes the hardware resource either by partitioning or sharing real hardware resources, or by emulating their behavior. The runtime, which CP offers to the guest operating systems, is called the “virtual machine.” It has its own memory, devices, and processors, which can be either real or virtual. The z/VM virtualization technology allows you to run several copies of the same or different operating systems on one zSeries server at the same time.

The configuration parameters of the z/VM virtual machines (guests or users) are stored in the so-called CP directory (USER DIRECT). The resources available to the guests (CPU, DASD, network adapter) are specified in this file. z/VM virtual machines, guests, or z/VM userids are all basically the same: each means a runtime environment for a zSeries operating system.

CMS is an operating system that runs in a virtual machine and offers an end-user interface and a development interface like a UNIX shell environment.

A special z/VM disk type is the “minidisk.” Minidisks are virtual disk devices; they can be either a part of or a complete DASD. Minidisks can be shared among several guest systems. A real disk can also be dedicated to a virtual machine.

The communication between the guest systems and z/VM can be established with point-to-point connections. Channel-to-channel (CTC) connections can be either real zSeries channels or virtual channels emulated by z/VM. Inter-User-Communication Vehicle (IUCV) is a virtual communication link that doesn’t use channels: the data transfer is only in main memory and no I/O hardware is involved. A third method of interconnecting guest systems within z/VM is provided by guest LANs, which are simulated Hipersockets and are described in detail in the redbook *zSeries HiperSockets,* SG24-6816.

While z/VM is a very stable operating system, factors such as human error, hardware failure, planned outages, and so forth make it impossible to guarantee that the system is 100% available. If the system hosts a large number of Linux images, a terminating z/VM leads to all images within z/VM being affected.

Error recovery in z/VM

The zSeries microcode and z/VM try to recover most errors, including such things as intermittent and permanent machine errors and system I/O errors, without manual intervention.

When an error occurs, CP records it and sends a message to your z/VM primary system console. The message notifies you of the error and tells you whether:

- System operation can continue.
- System operation can continue, but with fewer resources.
- System restart and recovery is beginning.

If the error condition doesn’t damage system integrity and can be assigned to a single guest, z/VM makes a soft abend of the guest and operation continues. This means that in almost all cases a crash of a guest system doesn’t affect CP operation, and other guests can operate without disruption.

In all cases a message is send to the system console. This message can be analyzed and actions started according to the error condition.

The zSeries hardware is able to detect CPU errors and transparently switch to another processor for continuous operation (CPU sparing). This function is transparent to the operating system, such as z/VM, z/OS and Linux for zSeries.

Automation of Linux guest start and stop process

During z/VM takeover, an important procedure is the effective start and stop process of all Linux guests.

Automated start procedure

For regular z/VM system start (IPL), or after a z/VM termination, an automated restart of z/VM can be configured. During system initialization, CP automatically logs on (starts) user AUTOLOG1. AUTOLOG1 can automatically start other virtual machines when issuing the XAUTOLOG command in the PROFILE EXEC of AUTOLOG1.

If a certain start order is necessary due to application dependencies, you can implement a startup procedure for the guest systems. This startup procedure is called within the PROFILE EXEC of user AUTOLOG1. The procedure is written in the script language REXX. An example of such a REXX procedure is given in Example 1.

Example 1 REXX procedure to automatically start two Linux guest systems

```
DEMOSTRT EXEC:
/*trace i */
address command cp xautolog linux1
address command cp sleep 9 sec
address command cp xautolog linux2
```

With this short REXX procedure, two guests (linux1 and linux2) will be started. After the start of the first guest, the system waits for 9 seconds before the second guest will be started. To improve performance by avoiding the search for execs or CP commands, simply include the instruction ADDRESS COMMAND at the front of your command. For more information on REXX see *z/VM REXX/VM Users Guide,* SC24-5962 and *z/VM REXX/VM Reference,* SC24-6035.

A guest can be configured to start either with CMS or with the Linux operating system. If starting with CMS, you can start the Linux operating system with the PROFILE EXEC of the guest. This indirect start of Linux has the advantage that some CP commands to prepare the environment can be issued.

The startup time of z/VM with Linux guest systems depends on the available hardware resource (CPU, memory, I/O channel), the number of Linux servers, and the application initialization time.

To avoid long Linux file systems check times, the use of a journaled file system such as ext3 is recommended. If the total amount of virtual storage of all the Linux guests exceeds the main storage size, it is recommended that you not start all guests at the same time.

Automated stop procedure

If a site takeover of the whole z/VM system with all guest systems must be performed, due to a planned or an unplanned outage, it is easiest to use a procedure to stop all guest systems. This stop procedure can be implemented with a REXX procedure shown in Example 2.

Example 2   REXX procedures to stop Linux guests

```
DEMOSTOP EXEC:
------------------------------------------------------------

trace i
say 'stopping Linux systems
say '       ********             '
say '                            '
'cpsend linux1 root '
'cp sleep 2 sec'
'cpsend linux1 passw0rd '
'cp sleep 2 sec'
'cpsend linux1 halt '
'cp sleep 2 sec'
say 'now wait 60 seconds for linux to stop'
say 'then logoff the systems              '
say '       ********             '
say '                            '
'cp send linux1 logoff'

CPSEND EXEC:
---------------------------------------------------------------------

/* REXX EXEC to send console input to a Linux virtual machine */
parse arg linuxid cmd
upper linuxid
cpcmd = 'SEND' linuxid cmd
cprc = substr(Diagrc(8,cpcmd),1,9)
exit cprc
```

Linux is case sensitive; thus the commands send to Linux must be in mixed case. The REXX function DIAGRC must be used to send commands in mixed case. In Example 2 the REXX procedure CPSEND implements the send functionally.

The problem with this procedure is that a password in clear text must be supplied. To avoid the use of a password in the REXX procedure, you can start the Linux guest with a so-called “open Linux console” in z/VM. To allow Linux to accept commands sent directly with the z/VM CP SEND command, you can replace the following line in the /etc/initab file of the Linux guest:

```
   1:1235:respawn:/sbin/mingetty console
```

with:

```
   1:1235:respawn:/bin/bash
```

The Linux server now accepts commands send with the CPSEND REXX procedure shown in Example 2. The Linux console is still protected by the userid and password of the z/VM guest machine, and there is no need to keep the Linux root password within a REXX procedure any longer.

Communication between z/VM systems

The heartbeat mechanism is an important component of a high availability solution and thus a reliable connection between two z/VM systems must be established. Two or more zSeries servers can be connected with a channel-to-channel (CTC) connection over ESCON or FICON. Enterprise Systems Connection (ESCON) and Fiber Connection (FICON) are zSeries-specific I/O architectures. The CTC connection simulates an I/O device for z/VM and provides the data path and synchronization for data transfer between two channels (I/O ports). With a CTC connection, a loosely coupled multiprocessing system is established.

Communication between the z/VM systems requires a communication program that handles the message transfer. The Remote Spooling Communications Subsystem (RSCS) is a networking program that enables users on one system to send messages, file commands, and jobs to other users on a remote system.

RSCS supports several transport protocols:

- Non-SNA
- SNA
- TCP/IP

With RSCS we are able to build up a network of z/VM systems that use the reliable CTC connections between zSeries systems.

More information on RSCS can be found in the RSCS document *Virtual Machine Remote Spooling Communications Subsystem Networking General Information,* GH24-5218

The Remote Execution Protocol (REXEC) server is a daemon that executes commands from a remote client. The REXEC client sends a userid, password, and the command to the remote host and receives the results. The REXEC server is part of the TCP/IP of z/VM. You can use it to build up high availability features. For more information about REXEC, see *z/VM TCP/IP Level 420 Planning and Customization,* SC24-6019.

Programmable Operator Facility (PROP)

One component of a high availability solution is the monitoring tool that monitors the hardware as well as the software components. One tool available for monitoring in z/VM is the “PROP” program.

In the z/VM environment, operating system components like CP, CMS, and even virtual machines generate messages that contain information about the status and availability of hardware and/or operating system components. System messages can be sent to a single z/VM “system operator.” Virtual machines can be configured to send their messages to the system operator as well.

The Linux console messages can be send to the PROP when specifying the PROP user as the secondary user of the Linux virtual guest. This allows you to integrate the Linux messages in the PROP automation feature.

The secondary user OP1 is assigned to receive the message from the Linux guest if you code the following statement in a Linux guest user directory:

```
CONSOLE 009 3270 T OP1
```

You can also set or change the secondary userid with the `set secuser` CP command.

PROP was developed to increase the manageability of the system and to allow remote operation of systems in a distributed environment. PROP intercepts all messages and requests directed to its assigned virtual machine and handles them according to predefined actions. In principle, this functionality allows you to monitor error messages from a failing component and start predefined actions, for example, send a message to the Linux system and /or initiate the restart of a service.

PROP uses a routing table to get the information needed to analyze the messages and start an action routine. Action routines are programs or execs that receive control in response to the match of a message and a routing table entry. They handle a particular type of message or command intercepted by the programmable operator facility. Action routines can be either REXX routines or PROP-supplied action routines. One of the these PROP-supplied routines, DSMPOR allows for issuing z/VM commands.

PROP can be configured for a distributed environment with several LPARs on the same and/or different zSeries servers. The z/VM systems must be connected with the RSCS facility of z/VM.

Example of using PROP

Figure 3 shows a sample configuration used for our PROP tests. In the illustrated scenario the Linux guest LNX31 sends its messages to guest OP1. PROP is started in guest OP1 and the messages send by LNX31 are processed. PROP is configured to react on “Kernel panic” messages and call the REXX procedure FORCELNX. The FORCELNX procedure is shown in Example 3 on page 15.

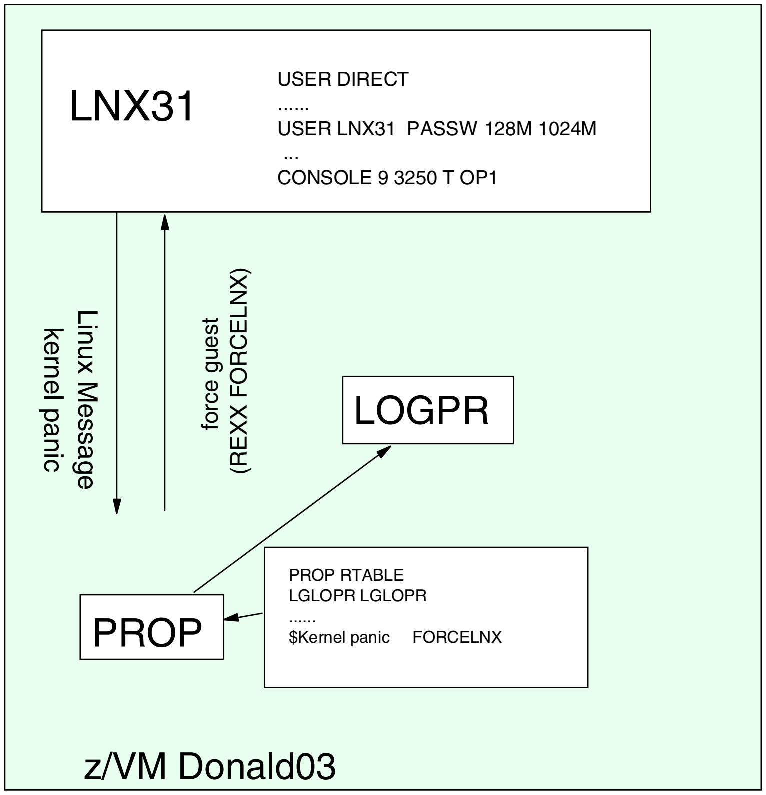

Figure 3 PROP configuration and message flow

The first parameter is the guest userid passed from PROP. In this example we definitely want to stop a guest in case of a severe kernel problem. The routing table for this scenario is shown in Example 4 on page 16. The message flow is presented in Figure 3. The kernel panic message is send to the OP1 user where the PROP is started. In this sample the secondary user of the LNX31 guest is specified in the user directory with the CONSOLE statement. PROP finds a match in the routing table and calls the REXX procedure FORCELNX. This procedure sends a CP **force** command to stop the guest system.

Example 3   REXX procedure FORCELNX

```
/* */
 PARSE UPPER ARG  USER_ID .
'CP FORCE ' USER_ID
```

The first parameter PROP transfers to the action routine is the source userid. In the case of a remote guest this userid is the RSCS userid. The userid of the remote guest is contained in

Example 4   Example of a PROP routing table

```
*        ----- SPECIFY THE PROP CONFIGURATION -----
* IDENTIFY THE LOGICAL OPERATOR
LGLOPR LGLOPR
*LGLOPR OPERATOR HOSTNODE
* BLANK SEPARATOR IS '/', ARBCHAR SEPARATOR IS '$', "NOT" SYMBOL IS '^'
TEXTSYM / $ ^

* DO LOGGING WHEN THIS TABLE IS IN EFFECT

LOGGING ON

ROUTE    ----- END OF CONFIGURATION SPECIFICATION -----

*----------------------- --- --- -- -------- -------- -------- --------
*T                        S   E   T U        N        A        P
*E                        C   C   Y S        O        C        A
*X                        O   O   P E        D        T        R
*T                        L   L   E R        E        N        M
*----------------------- --- --- -- -------- -------- -------- --------
* FILTER OUT LOGON, LOGOFF, ETC. MESSAGES SO OPERATOR NEEDN'T SEE THEM
*----------------------- --- --- -- -------- -------- -------- --------
/OUTPUT OF                19  27  3
/LOGON                    19  23  3
/LOGOFF$^FORCED           19  80  3
/DSCONNECT                19  27  3
/RECONNECT                19  27  3
/DIAL                     19  22  3
/DROP                     19  22  3
$Kernel panic                                         FORCELNX
/FROM DONALD04(TCPMAINT):           RSCS     DONALD03 TESTHUGO
*----------------------- --- --- -- -------- -------- -------- --------
* SEND FILTERED ASYNCHRONOUS CP MESSAGE STREAM TO LOGICAL OPERATOR
*----------------------- --- --- -- -------- -------- -------- --------
                                  3                   DMSPOS   LGLOPR
*----------------------- --- --- -- -------- -------- -------- --------
* SEND A CP OR CMS COMMAND TO VM TO BE EXECUTED
*----------------------- --- --- -- -------- -------- -------- --------
/CMD /                     1   4    OPERATOR HOSTNODE DMSPOR   TOVM
/CMD /                     1   4    OPERATOR PROPNODE DMSPOR   TOVM
.....
```

Cross System Extention (CSE)

A failure of z/VM can be regarded like a hardware failure. One method to handle hardware failures is to use a cluster. In a cluster scenario several systems are connected together and handle the workload. In case of a failure of one system in the cluster the other systems take over the load of the failing system. A limited cluster technology exists for z/VM: the Cross System Extensions (CSE). This technology is designed to support a shared z/VM environment. It provides synchronization of the user directories and spool files of the z/VM systems in the CSE complex. The problem in this environment is that we have no direct CSE support for managing a guest failure. This means that we have to develop procedures to handle a failure of a Linux guest system. These procedures are similar to the procedures that must be implemented in a non-CSE environment. The primary focus of CSE is the takeover of z/VM and z/VM applications. For these reasons the use of CSE technology to implement high availability for Linux guest systems is an unreliable approach in our scenario.

z/VM high availability

In this section the currently possible scenarios for z/VM systems are discussed. In the situation that only a few guests need high availability, these special guests can be integrated in a Linux high availability solution as described later in this paper.

If high availability with minimal downtime is absolutely necessary for all guest systems, one has to implement a hot standby scenario. The backup z/VM system can reside in a separate z/VM image in a LPAR on the same or on another zSeries machine.

High availability of z/VM with z/OS disaster recovery techniques

The design principles of a disaster recovery solution can be adapted to implement a z/VM high availability solution. Thus, the question arises whether the zSeries disaster recovery techniques can be used for a z/VM environment in conjunction with z/OS. Key technology for implementing a disaster recovery solution with zSeries and z/OS is the Geographically Dispersed Parallel Sysplex (GDPS).

GDPS

GDPS is a multi-site application availability solution that provides the ability to manage the remote copy configuration and storage subsystems, automates Parallel Sysplex operational tasks, and performs failure recovery from a single point of control. For more information see redbook *OS/390 MVS Parallel Sysplex Configuration Volume 1: Overview*, SG24-2075. The following technologies are used by GDPS:

- Parallel Sysplex
- System Automation for OS/390
- Enterprise Storage Subsystems (ESS)
- Peer-to-Peer Virtual Tape Server (PtP VTS)
- Optical Dense Wavelength Division Multiplexer(DWDM) and key IBM open architectures:
  - PPRC (Peer-to-Peer Remote Copy) architecture
  - XRC (Extended Remote Copy) architecture
  - Virtual Tape Server Remote Copy architecture

GDPS provides switching capability from one site to another site, for planned and unplanned outages. The latest version of GDPS can manage zSeries images that execute externally to the Parallel Sysplex. This support offers only a restart of the LPAR in which the z/VM environment runs and the PPRC management.

For more information, see the IBM white paper *Geographically Dispersed Parallel Sysplex: The Ultimate e-business Availability Solution,* GF22-5114-02.

Failover of z/VM and guest systems with GDPS techniques

In a mixed z/VM and z/OS environment, failover by means of GDPS can be implemented. This section describes how disaster recovery for z/VM and Linux for zSeries guest systems can be implemented. In the case of an outage of the primary z/VM system due to a processor, storage subsystem, or site failure, this configuration enables you to start another z/VM with the Linux guests and give access to the same data and services.

In this disaster recovery configuration the guest images on both sites, the primary and the secondary, access the data from their local storage subsystems, which are kept in sync by PPRC. This ensures that in case of a hardware or site failure the secondary site can take over the data and services. This makes it necessary that even in a disaster case where the primary DASD is not involved, a site switch of these volumes has to be performed.

In most cases the Linux guests in the disaster recovery case will take over all network resources (IP address, host names, routing table, and so forth) depending on the installed network infrastructure. In this situation the network configuration can be stored on minidisks that are copied from the primary to the secondary site using PPRC. If a different network setup is required for the recovery site, you have to create separate configuration files for each location, which are placed and maintained at both sites.

The Linux guests should be defined with the same device addresses for the primary and the secondary site.

Currently there is no interface between the GDPS and z/VM or the Linux guest systems under z/VM. But GDPS-controlled z/OS systems, which are running on the same Central Processor Complex (CPC) and/or are attached to the same storage subsystems as the z/VM with the Linux guests, can pass the failure conditions of the devices to the GDPS.

In case of a failure of a critical resource, like a storage subsystem, the failover procedure for the z/VM with the Linux guests is triggered and following actions are performed:

1. Reset the z/VM LPAR.
2. Break the PPRC DASD pairs.
3. Load the z/VM system on the secondary site, with automatic startup of all Linux guest systems.

These very critical actions are performed automatically, without any operator interaction.

PPRC ensures that all data, which is written on the primary site, is also written in the recovery site. But this does not ensure data consistency on the application level; this must be handled by the application software.

GDPS can manage LPARs with z/VM, but GDPS has no awareness of whether the z/VM with the Linux guests is functional or not. GDPS uses Parallel Sysplex communications (XCF) to verify that all the z/OS in the Sysplex/GDPS cluster are functional and, if an image fails, to either prompt the operator for permission to re-IPL the image, or automatically re-IPL the image. Special communications between the z/VM system and GDPS must be developed to integrate the z/VM error messages into the Sysplex/GDPS cluster.

z/VM site takeover

In some cases a cold standby scenario is applicable, especially if the service level agreement allows for certain restart times or if you have a planned outage due to maintenance. The purpose of this section is to give a rough idea of how a z/VM takeover might look. It is out of the scope of this paper to describe a full takeover procedure.

If the outage is going to be brief, the best procedure is to take no takeover action: when the system has a properly developed start procedure, the z/VM systems restarts automatically. A takeover must be performed only in cases where a hardware failure prevents the system from restarting.

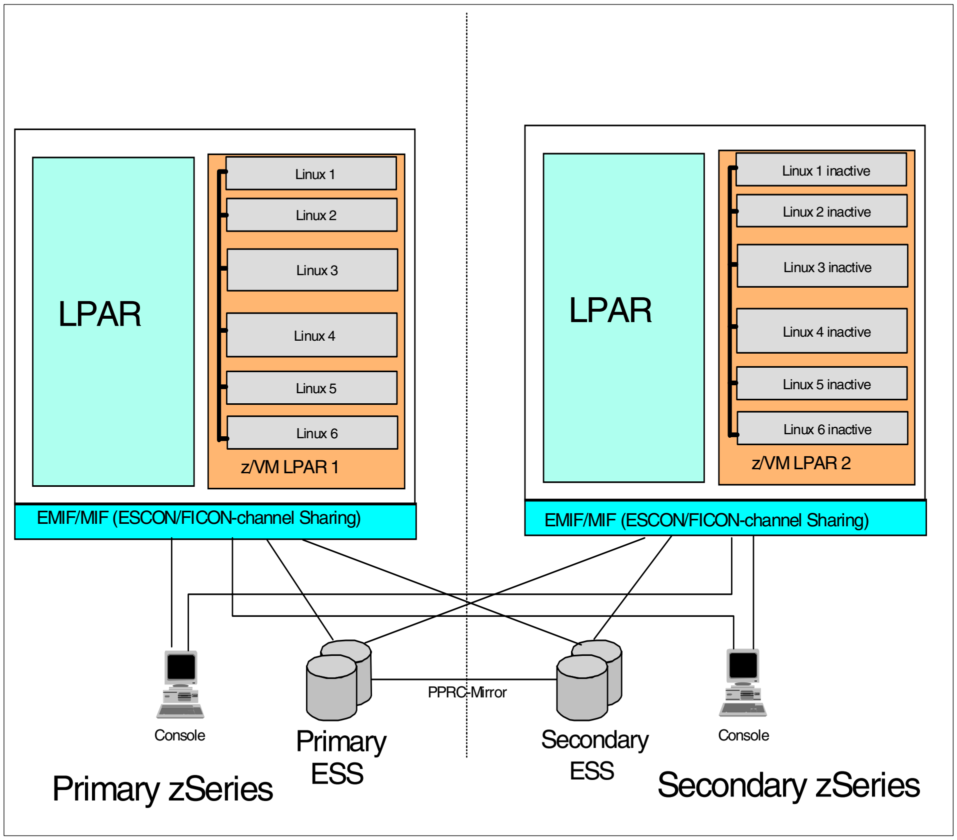

Figure 4 z/VM high availability scenario cold standby

If the failure of z/VM is serious and a restart isn't successful, a z/VM takeover must be performed. Two LPARs with z/VM must be configured to prepare for a z/VM takeover. The LPARs can be on the same or on two different servers, depending on the high availability requirements you have. If the zSeries hardware reliability features are sufficient according to your service level agreements, you can implement the second z/VM LPAR in the same zSeries server.

The design principles of a z/VM takeover can be adapted from the disaster recovery scenario with GDPS. Linux and z/VM can not directly participate in a Parallel Sysplex, and Linux has no direct support for PPRC. This means that you have to develop a procedure to make a site takeover. More information on implementing PPRC with z/VM can be found in the redbook *Implementing ESS Copy Services on S/390,* SG24-5680.

Figure 4 shows a configuration suitable for a z/VM takeover with two zSeries servers. In this figure only the z/VM Linux LPARs that participate in the takeover are shown. Clearly, other LPARS on the zSeries server can be used to run other workload. In addition, two disk subsystems with PPRC as data replication techniques are used. It is important in an error situation to install a second hardware management console.

The z/VM LPAR must be configured on both sides with all the connections to the network and disk subsystems. The start of the LPAR and the z/VM system can only be performed with the hardware management console (HMC). To make the takeover procedure easier, it is possible to activate the second LPAR and start the z/VM system without starting the Linux guest systems. In this case an automated start of the Linux guest systems can be initiated with a REXX startup procedure and the XAUTOLOG command of z/VM. To start this procedure remotely the REXEC or RSCS with PROP techniques can be used.

A hardware failure can cause z/VM to abend. If the z/VM resident system volume, for example, is not accessible due to a control unit failure, z/VM can't recover the error and abends. In this case the mirrored data must be made available. To do this the devices from the primary control unit must be set offline and the devices of the secondary (backup) control unit have to be set online. The backup devices are mirrored versions of the primary ones and contain the same data. The data are mirrored with PPRC or other techniques, like network block devices in combination with software RAID. After the data are available to the secondary system, z/VM and the Linux guest systems can be initiated. The PPRC management must be done with ICKDSF commands or with the PPRC Web interface. Unfortunately, no service like GDPS is available for a z/VM-only environment.

A complete zSeries server failure activates a complete site takeover. With a PPRC solution implemented, the secondary system can be activated with the same data as the primary system. The network connection can be resumed without a problem and the same network addresses can be used as before. The zSeries Open Systems Adapter (OSA) makes itself known to the network, thus the hardware change is transparent to the clients in the network.

To decrease restart times and to improve manageability, you would have an activated LPAR and a running z/VM on the secondary site. Then, in case of a takeover without further PPRC actions, you are able to start the Linux guests with the technique described.

More information about z/VM high availability choices can be found in the redbook *Linux for zSeries and S/390: ISP/ASP Solutions*, SG24-6299.

General topics for Linux and z/VM

Some specific procedures and techniques are needed to build a high availability solution involving both z/VM and Linux. This section addresses some general basic technologies useful in building a high availability solution for Linux and z/VM.

How to handle an Open System Adapter (OSA) failure

In general, to avoid a single point of failure, redundant hardware is required. This is especially true for the OSA adapter. A failure of the OSA adapter disconnects all Linux guests from the network.

With Virtual IP Addressing (VIPA) it is possible to build a fault-tolerant network attachment. The benefit of VIPA is that the TCP/IP host is freed from dependence on particular network attachments. If the physical network device fails, the network recovery can be done without manual network reconfigurations. VIPA works only in conjunction with dynamic routing within z/VM.

More information on z/VM VIPA configuration and other failover scenarios (for example, a second z/VM TCP/IP stack), can be found in the redbook *TCP/IP Solutions for VM/ESA* SG24-5459.

Figure 5 on page 22 shows a sample z/VM and Linux configuration. In this sample two OSA Express adapters are dedicated to the z/VM TCP/IP stack. A virtual IP address located in a different subnetwork from the real IP addresses of the OSA adapters is specified. The z/VM guests can be connected with a point-to-point connection (CTC or IUCV) or with a guest LAN. The IUCV point-to-point connection is preferred to the CTC connection because the first needs fewer resources and, in case of a failover, no COUPLE command must be issued. In our sample the guests are connected with CTC connections. Also the CTC connections are in a different subnet from the VIPA, and the z/VM TCP/IP stack is the router for the guest systems.

A sample PROFILE TCPIP file for this configuration is given in Example 5 on page 23. In the HOME statement the virtual, real, and point-to-point IP-address are specified. The GATEWAY statement is used to specify the IP-addresses of the Linux guests.

The OSA Express adapter builds the OSA address table from the HOME statement of the z/VM TCPIP stack or directly connected guests. Thus the IP-addresses of the guest systems are stored in the OSA Address Table (OAT). To ensure that the IP-packets are sent to the guest systems, the OSA adapters must be configured as primary routers. For more detail on this see the redbook *OSA-Express Implementation Guide, SG24-5948*.

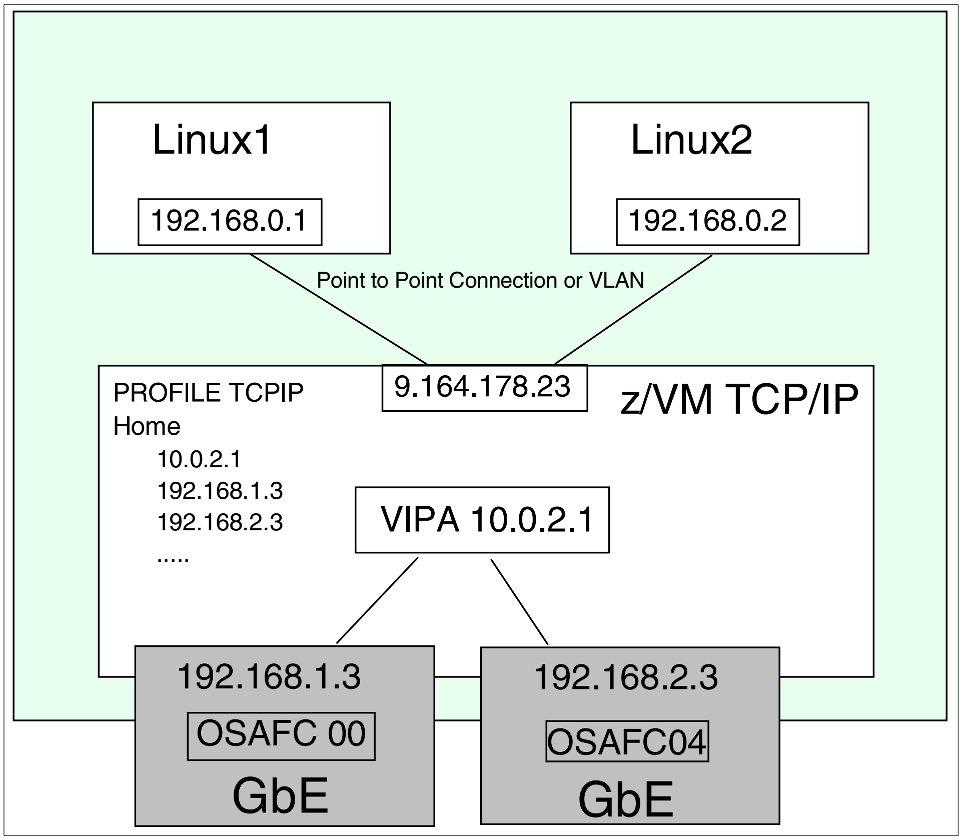

Figure 5 VIPA configuration with z/VM as TCP/IP router and Linux guest systems

Example 5   Sample PROFILE TCPIP

```
...
OBEY
  VMADMIN MAINT TCPMAINT OPERATOR REXEC MPROUTE
ENDOBEY
AUTOLOG
MPROUTE      PASSWORD          ;Multiple Protocol Routing  SERVER
ENDAUTOLOG

PORT
  23     TCP  INTCLIEN
  512    TCP  REXECD          ; REXECD  SERVER (REXEC)
  514    TCP  REXECD          ; REXECD  SERVER  (RSH)

INTERNALCLIENTPARMS
  ASYNCHRONOUSINPUT
  CONNECTEXIT  SCEXIT
ENDINTERNALCLIENTPARMS

ASSORTEDPARMS
; NOSOURCEVIPA
SOURCEVIPA
VARSUBNETTING
IGNORERedirect
ENDASSORTEDPARMS

DEVICE OSAFC00  OSD   FC00  PORTNAME GBFC00  PRIROUTER  AUTORESTART
LINK OSAFC00  QDIOETHERNET  OSAFC00
DEVICE OSAFC04  OSD   FC04  PORTNAME GBFC00  PRIROUTER  AUTORESTART
LINK OSAFC04  QDIOETHERNET  OSAFC04
DEVICE  VLINK1 VIRTUAL 0
LINK VLINK1 VIRTUAL  0 VLINK1
DEVICE LINUX1-CTC CTC   0610
LINK LINUX1-CTC  CTC    LINUX1-CTC
DEVICE LINUX2-CTC CTC   0620
LINK LINUX2-CTC  CTC    LINUX2-CTC

HOME
   10.0.2.1     VLINK1
   192.168.1.3  OSAFC00
   192.168.2.3  OSAFC04
   192.168.178.23 LINUX1-CTC
   192.168.178.23 LINUX2-CTC
; ENDHOME

GATEWAY
  192.168.1.0          =  OSAFC00        1492 0
  192.168.2.0          =  OSAFC00        1492 0
  192.168.0.1          =  LINUX1-CTC     1492 HOST
  192.168.0.2          =  LINUX2-CTC     1492 HOST
; ENDGATEWAY

START OSAFC00
START OSAFC04
START LINUX1-CTC
START LINUX2-CTC
```

To allow for automated OSA adapter failover we used, in this example, the Open Shortest Path First (OSPF) protocol. The MPROUTE server of the z/VM TCP/IP implements this dynamic routing protocol. The Multiple Protocol Routing (MPROUTE) server configuration file (MPROUTE CONFIG) is given in Example 6. The interfaces to the Linux guests and to the OSA adapters are specified. The static routes specified in the GATEWAY statement were dynamically updated through the OSPF protocol. For more information see *z/VM TCP/IP Level 420 Planning and Customization,* SC24-6019.

In case of an OSA adapter failure, the z/VM router recognizes the failure and sends an OSPF message to inform the router in the network segment that the VIPA is now available over the second adapter. The VIPA is not visible to outside routers, thus a route to the VIPA must be configured on these routers.

Example 6   Sample MPROUTE CONFIG file

```
Area
   Area_Number=0.0.0.0
    Authentication_Type=None;
OSPF_Interface
      IP_Address=9.164.178.23
      Name=LINUX1-CTC
      Subnet_Mask=255.255.255.255
      Destination_Addr=192.168.0.1
      Attaches_To_Area=.0.0.0.0
      Cost0=5;
OSPF_Interface
      IP_Address=9.164.178.23
      Name=LINUX2-CTC
      Subnet_Mask=255.255.255.255
      Destination_Addr=192.168.0.2
      Attaches_To_Area=.0.0.0.0
      Cost0=5;
OSPF_Interface
      IP_Address=192.168.1.3
      Name=GBFC00
      Subnet_Mask=255.255.255.0
      Attaches_To_Area=.0.0.0.0
      Cost0=2;
      Router_Priority=1;

OSPF_Interface
      IP_Address=192.168.2.3
      Name=GBFC04
      Subnet_Mask=255.255.255.0
      Attaches_To_Area=.0.0.0.0
      Cost0=2;
      Router_Priority=1;
OSPF_Interface
      IP_Address=10.0.2.1
      Name=VLINK1
      Subnet_Mask=255.255.255.0
      Attaches_To_Area=.0.0.0.0
.....
```

Another possibility to implement a network interface failover is to connect each Linux guest with two adapters and assign two IP addresses, a service and a standby address, to each Linux guest. If a network adapter fails, the Linux guest is reachable through the second adapter. Again a dynamic Linux router will be used to inform the other routers in the net that a new route to the Linux system is active. For a detailed description of this scenario see “High availability test scenarios” on page 42.

DASD sharing

Data sharing between the service and the backup node in a high availability cluster is a key issue. From the hardware point of view, the system administrator has to configure the hardware I/O on both nodes to have access to the same DASD. The zSeries I/O configuration is stored in the I/O Configuration Data Set (IOCDS). In the IOCDS the system administrator must configure paths to the disk on the subsystem.

Not only must the hardware support shared DASD, the operating systems also have to provide features for DASD sharing. DASD sharing is supported in z/VM. Only full pack minidisks can be used to share DASD. A full pack minidisk is a virtual disk that is defined over an entire real device.

In the Linux guest configuration (USER DIRECT) you have to code an MDISK statement. In our case, we used the following MDISK statement:

```
MDISK 0250 3390 0001 3338 FR5218 MW READ WRITE MULTIPLE
```

This defines a minidisk with device number 250. This MDISK statement is the same in the Linux guest configurations on both sites.

Shared read/write access to more than one guest system can be configured, but z/VM doesn't care about the data consistency. Data integrity can only be achieved if the Linux file system supports shared read/write access.

In a cold standby scenario it is possible to configure the DASD as shared read/write accessible but not mounted on the backup node. In a failover situation the file system must be unmounted from the primary node and mounted to the secondary. Precautions must be taken so that both nodes don't have access to the data at the same time.

CP is able to cache minidisks in expanded or main memory to increase I/O performance. If minidisk caching (MDC) is on, you can have problems with read/write operations on shared DASDs. An update to a DASD MDC on one site caches the data only on that site; therefore, the sites have different data in their cache and thus data integrity is not maintained on both sites. For this reason it is recommended that you switch off the minidisk cache.

For hot standby scenarios it is not possible to ensure data consistency with the methods described in this section. Data consistency can only be guaranteed from the application layer or with special file systems.

File systems

One major issue in a highly available environment is that the data must be available for all nodes in the cluster.

For short restart times a journaling file system is recommended. Some of the file systems ported to the Linux for zSeries platform are delivered with Linux distributions; for example, ReiserFS for SuSE and ext3 for Redhat. Table 2 lists the available journaling file systems for Linux on zSeries, along with their kernel dependencies and the distribution in which they are contained.

Table 2 Available journaling file systems for Linux on zSeries

File system	Kernel	Distribution
ext3	2.4.7	RedHat 7.2
reiserfs	2.4.x	SuSE SLES 7
jfs	2.2.14 2.4.16	SuSE SLES 7 (refresh)

A hot standby scenario with read/write access to the data needs a distributed file system (ideally in combination with a journaling feature). Several distributed file systems exist, each with its own strengths and weaknesses. It was beyond the scope of this paper to evaluate the available file systems in detail, so only a short summary is given.

One distributed file system is the network file system like NFS. NFS is simple to install and use with Linux. NFS is not able to recover from a system crash. In version 3 of NFS the central NFS file server is a single point of failure.

The Global File System (GFS) from Sistina is a shared file system that is designed for high availability solutions. GFS is not an open source project, so it was not used in this project. There is an ongoing OpenGFS project (`http://www.opengfs.org`). Support for Linux for zSeries was not yet available at the time of this writing.

Andrew File System (AFS) is a distributed network file system without a single point of failure, but the effort needed to set up and manage this file system is high.

With Samba, files can be made available for several clients. How a high availability scenario for a Samba server can be established is explained later in this document.

Table 3 lists the available distributed file systems. None of these files systems is included in a Linux for zSeries distribution at the present time.

Table 3 Distributed file systems

File system	Remarks
NFS3	Logging in memory, so no recovery after system crash NFS<4: single point of failure
NFS4	Logging in memory, so no recovery after system crash
GFS	Sistina; not open source
AFS	OpenAFS; no Kernel recompile - loadable module; fits needs of Web scenarios; high administration effort
SAMBA	Useful in combination with Windows clients, NFS preferred

Stonith

A partitioned cluster situation (see “High availability example” on page 6) can lead to damaged data. Damage to the data can be avoided by killing the primary node from the secondary node before the resources are transferred. This procedure is called STONITH (shoot the other node in the head). In a z/VM environment we have several possibilities to implement STONITH:

- Control guest
- REXEC server in z/VM
- Remote message to PROP

These methods are described, along with their advantages and disadvantages, in this section.

Control guest

For each site a so-called “control” Linux guest is configured and started. This guest has the authority to force other Linux guest systems. On both sites the control guest and the other guests are connected over a CTC or IUCV connection (see Figure 6). The guests can send a **`force`** command over ssh to the control guest, which performs this **`force`** command with the CPINT utility. CPINT is a utility for sending CP commands from Linux to CP. It is distributed within all Linux for zSeries distributions, or you can download the tool from the following Web site:

http://linuxvm.org/penguinvm/programs/

The authentication is made with RSA keys. Both sites are connected via TCP/IP.

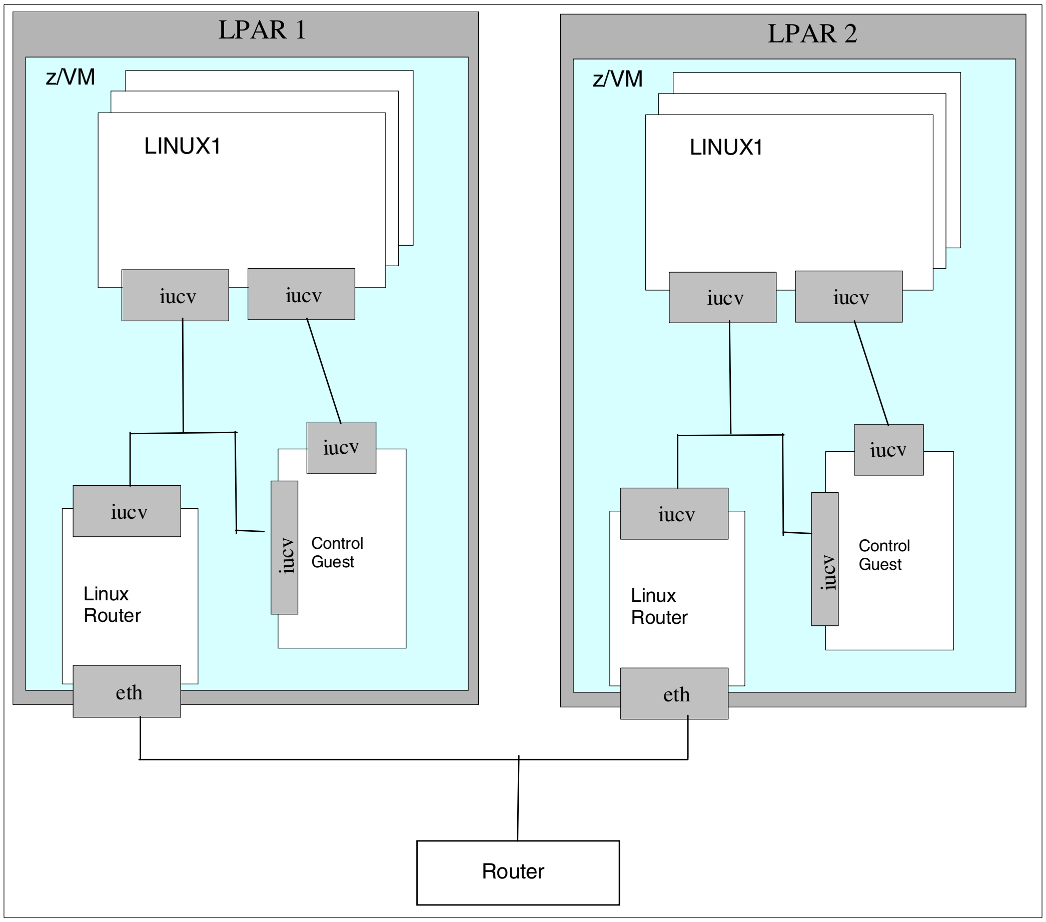

Figure 6 Stonith with Linux control guest

Advantages: Secure method
Simple implementation
Linux-only solution
No direct connect to z/VM
Good control of which guests are forced

Disadvantages: Failure of the control guest (single point of failure)
More memory usage as a z/VM solution
RSA handshakes slow performance

REXEC server in z/VM

A REXEC program is installed and started in z/VM that receives commands from the remote sites. An overview of this configuration is presented in Figure 7.

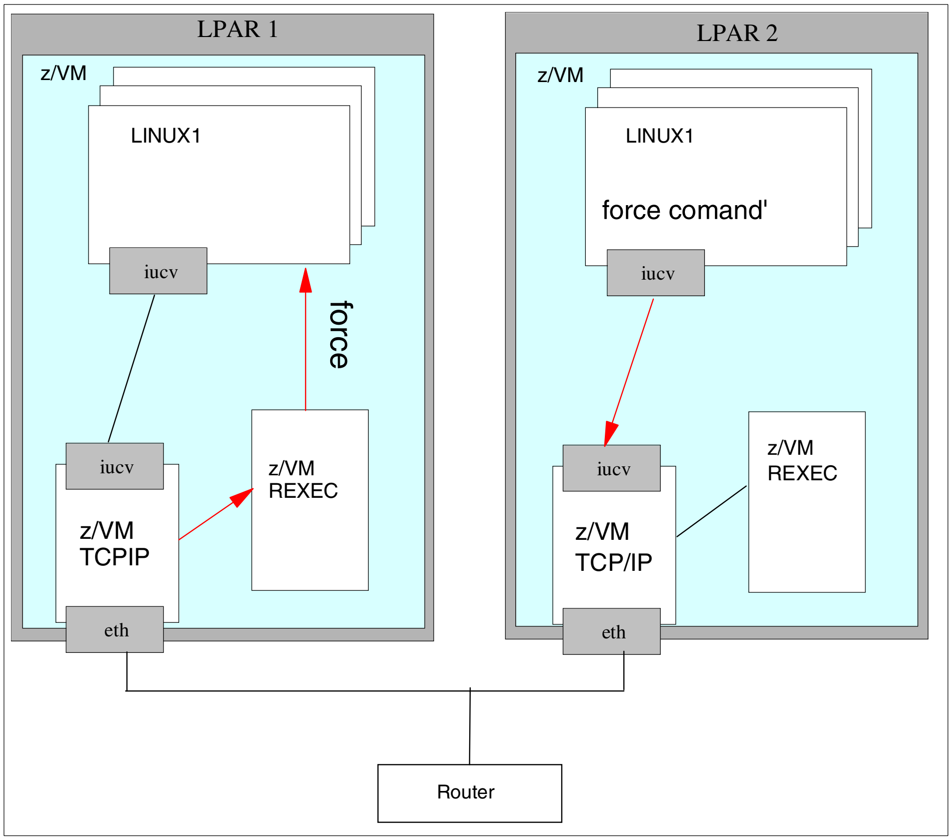

Figure 7   Stonith with REXEC server

Both sites must be connected over TCP/IP. The guest which will take over the resources sends the command to force the guest directly over the TCP/IP connection to the remote REXEC daemon. The REXEC executes the command.

Advantages: No extra Linux guest
Simple setup
Disadvantages: Failure of REXEC server
Unsecure connection from Linux to z/VM
No authentication

Remote message to PROP

PROP is configured and active on both sides. RSCS as the communication vehicle between both z/VM systems is configured and active. See Figure 8 on page 29 for an illustration of the configuration.

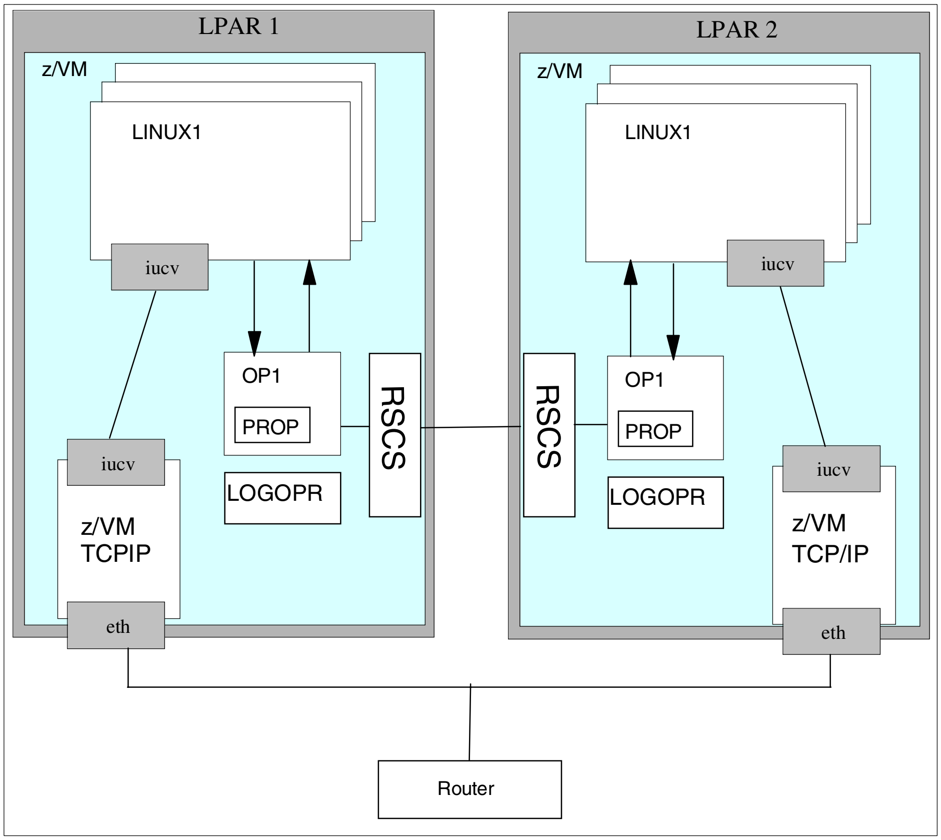

Figure 8 Stonith with PROP

Advantage: Simple usage
RSCS communication over separate connection

Disadvantage: RSCS is a priced z/VM feature

High availability of z/VM with network dispatcher

Network dispatcher can provide a high availability solution for a z/VM environment.

In a Web server scenario with static data (meaning the data can be shared read-only) the network dispatcher can be used to provide high availability functionality. If one z/VM system is down, the network dispatcher is not able to send requests to the Web server running on the Linux guests of this z/VM system and therefore redirects the requests to the remaining Linux guest in the other z/VM system. The network dispatcher can be implemented with software (WebSphere Edge Server) or with hardware (for example, CISCO CSS Router).

Another example where the network dispatcher can be used for high availability is a J2EE Web application server scenario. The application data are stored in a shared DB2 on z/OS with Parallel Sysplex and the application runs on Linux for zSeries. The application runs on all Linux servers. The network dispatcher detects a failing z/VM, HTTP server, and/or Web application server running in a Linux guest, and directs all following requests to the other Linux server. The Linux guests are distributed over different z/VM systems and zSeries machines. That means a hot standby scenario, as illustrated in Figure 9 on page 30. Important for this network dispatching scenario is that you have to implement a shared data environment for all nodes in the environment (such as highly available DB2 on z/OS).

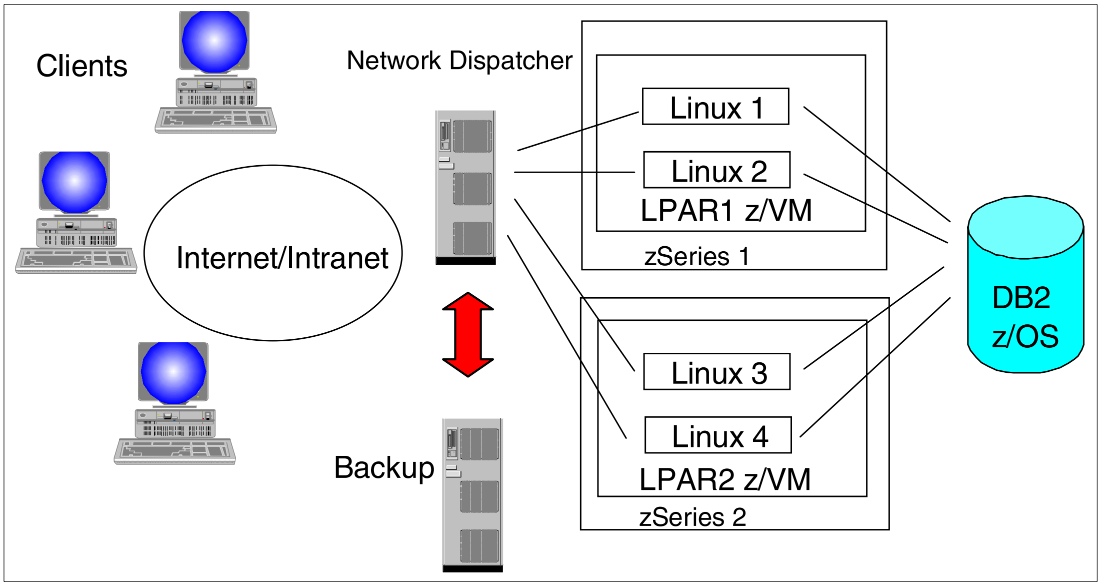

Figure 9   High availability scenario with network dispatcher

If shared data access is needed but it cannot be implemented for any reason, the network dispatcher alone can’t be used for high availability. Additional procedures for data and application takeover must be started.

Practical Linux high availability

So far we have discussed the general aspects of high availability and the role of z/VM in this effort. We now focus on the Linux operating system. The Open Source Linux High Availability Project is a good starting point for learning about issues related to developing highly available Linux solutions (see http://www.linux-ha.org). Unfortunately, the focus of this project is Linux/Intel. While most of the tools and deliverables can be easily ported to Linux for zSeries, they must be carefully checked to verify whether all components work as expected.

Our test setup

It is not possible to show all the high availability options you can implement with Linux. The objective of this paper is to describe the basic components to begin with as a base for a complex HA environment. Generally, for an HA solution, you have to implement application, data, network, and hardware takeover scenarios. We chose four simple scenarios which demonstrate the main issues and allow one to build up a more complex HA solution.

Hardware setup

To implement a Linux failover cluster with heartbeat we needed connectivity between the systems. For a connection to the outside world, we implemented by at least one separate connection, via OSA-2 cards, directly attached to the Linux guest systems. While the Ethernet can be used for heartbeats, it is preferable to use a medium which does not rely on IP. Therefore, we established CTC connections between the guests and used it as a medium for serial heartbeat.

Figure 10 on page 32 shows the general configuration of our 2 systems.

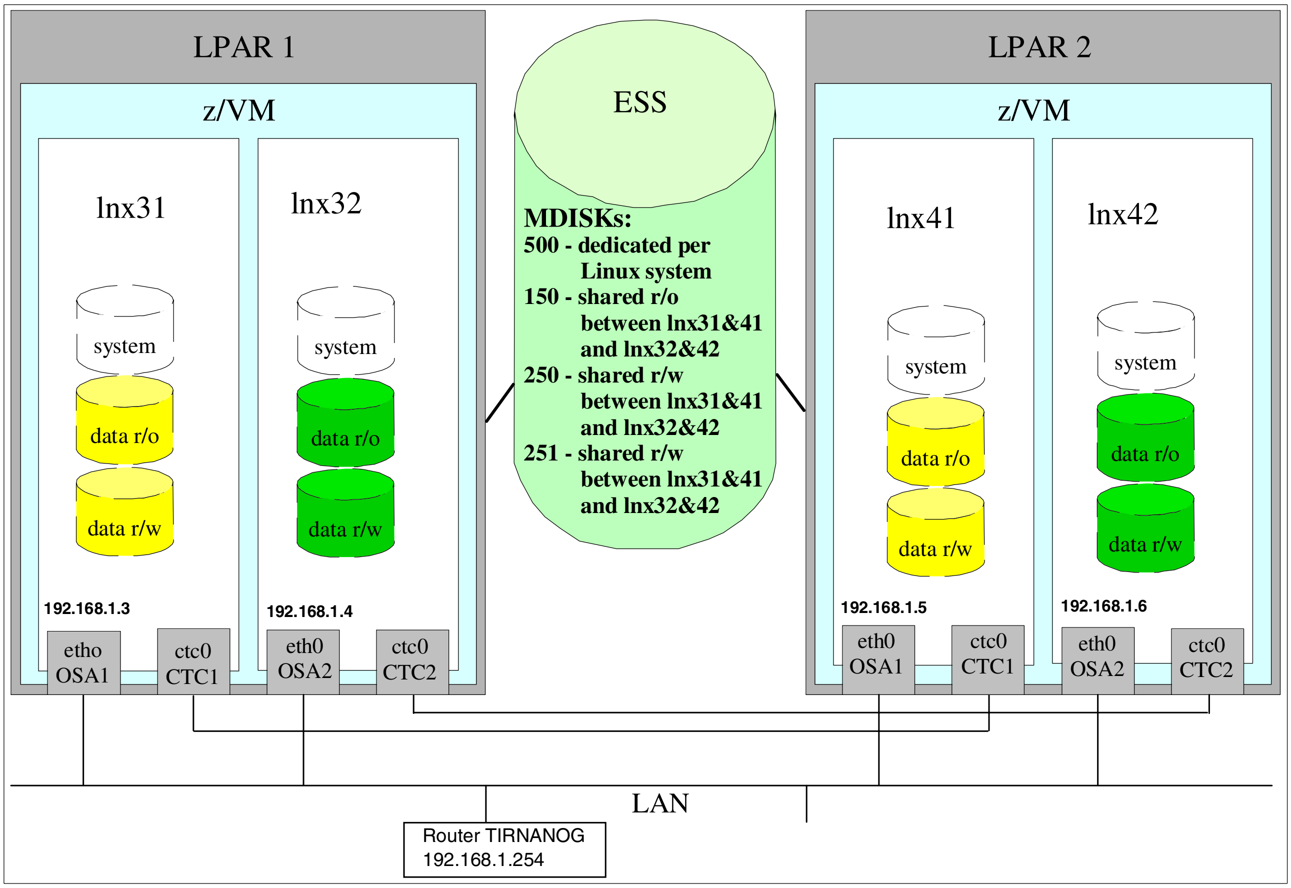

Figure 10 Hardware setup for the test environment

We had two pairs of Linux systems on which we performed the tests in different scenarios. Each scenario was done on only one pair, but the setup with two pairs allowed us to work in parallel. However, due to limited access to appropriate hardware, we used one zSeries with:

- 2 LPARs
- 2 OSA network adapter per LPAR
- 2 non-IP connection (CTCTTY) between the LPARs
- 1 dedicated DASD per Linux guest for the Linux system
- 1 shared read-only DASD (device 150) for all guests
- 2 shared read/write DASD (devices 250/251) for all guests

Setting up 2 machines would be the preferred way to implement a highly available solution because of possible hardware errors, planned maintenance outages, and disaster events on a single machine.

Software setup

We used the SuSE Linux Enterprise Server 7 (SLES 7) distribution in a default installation on one guest machine, and then used it as the clone basis for the other system. The additional packages we used are described in the following sections. The relevant software versions within SuSE SLES 7 are:

- Kernel 2.4.7
- glibc 2.2.2

- gcc 2.95.3
- ReiserFS 3.x.0

LVM

In the tests with data takeover we wanted to use the Logical Volume Manager (LVM). To configure the LVM only on the read/write shared DASD, we performed the following steps on virtual Linux server lnx31.

Before formatting the DASD, we checked if both DASD were online to lnx31:

```
lnx31# cat /proc/dasd/devices
```

The output shows that all necessary DASDs (250,251) are online (see Example 7).

Example 7 Output of the file /proc/dasd/devices

```
0500(ECKD) at ( 94:  0) is dasda:active at blocksize: 4096, 601020 blocks, 2347 MB
0150(ECKD) at ( 94:  4) is dasdb:active at blocksize: 4096, 600840 blocks, 2347 MB
0250(ECKD) at ( 94:  8) is dasdc:active at blocksize: 4096, 600840 blocks, 2347 MB
0251(ECKD) at ( 94: 12) is dasdd:active at blocksize: 4096, 600840 blocks, 2347 MB
```

Next we have to format the devices /dev/dasdc and /dev/dasdd:

```
lnx31# dasdfmt -b 4096 -p -f /dev/dasdc
lnx31# dasdfmt -b 4096 -p -f /dev/dasdd
```

To create a partition for the devices the **fdasd** command was issued:

```
lnx31# fdasd -a /dev/dasdc
lnx31# fdasd -a /dev/dasdd
```

The next step was the initialization of the physical volume:

```
lnx31# pvcreate /dev/dasdc1 /dev/dasdd1
```

The volume group was create with:

```
lnx31# vgcreate vg00 /dev/dasdc1 /dev/dasdd1
```

We wanted to use striping over the two read/write shared DASD with two stripes and a stripe size of 4 KB. The logical volume name was rw and the volume group name vg00:

```
lnx31# lvcreate -i 2 -I 4 -n rw -L 1G vg00
```

To back up the metadata of the newly created volume group, we issued:

```
lnx31# vgcfgbackup vg00
```

To reduce long Linux file system check times we used the journaling file system reiserfs. With **mkreiserfs** we created the file system.

```
lnx31# mkreiserfs /dev/vg00/rw
```

To check whether the configuration was successful, we mounted the newly created file system and copied some files to it:

```
lnx31# mount /dev/vg00/rw
```

An LVM start/stop script was created for the takeover (Example 8 on page 34).

Example 8 LVM start/stop script

```
#! /bin/sh
# Source SuSE config
. /etc/rc.config
# Determine the base and follow a runlevel link name.
base=${0##*/}
Link=${base#*[SK][0-9][0-9]}
. /etc/rc.status
case "$1" in
    start)
        echo -n "Starting LVM"
        pvscan
        vgscan
        vgchange -a y vg00

        # Remember status and be verbose
        rc_status -v
        ;;
    stop)
        echo -n "Shutting down LVM"
        vgchange -a n vg00

        # Remember status and be verbose
        rc_status -v
        ;;
    *)
        echo "Usage: $0 {start|stop}"
        exit 1
        ;;
esac
c_exit
```

heartbeat

For our HA tests we used the software “heartbeat,” which is available from:

http://linux-ha.org

You can either download the source in RPM format or as a gzipped tar archive. To compile and install heartbeat, the ucd-snmp package and the openssl-devel package must be installed first. While ucd-snmp is shipped with the SLES 7 distribution, the openssl-devel libraries only come with the developer version of SLES 7, which is available from SuSE. We used the latest version of heartbeat: 0.4.9.1.

We used rpm to build a binary package for deployment on our systems. To compile the package and create a binary rpm, you can either do it from the source rpm:

```
# rpm --rebuild heartbeat-0.4.9.1-1.src.rpm
```

Or if you use the tar file:

```
# rpm -tb heartbeat-0.4.9.1.tar.gz
```

This creates the binary package in /usr/src/packages/RPMS, which you then can install via rpm on every system in the cluster. After installation, you will find documentation in /usr/share/doc/packages/heartbeat/.

Since this paper only describes the specific settings needed for our scenarios, refer to the documentation (read GettingStarted.html) and online resources at http://linux-ha.org for general information on the heartbeat package.

To use heartbeat with SLES 7 you need to add a symbolic link in /etc to reflect the Red Hat-like directory structure which heartbeat was originally written for:

```
# ln -s /etc/init.d/ /etc/rc.d/init.d
```

Further, you need to edit the 3 configuration files to reflect your environment. The files are located in /etc/ha.d. Examples, which you can copy over to /etc/ha.d, are provided with the documentation. The files are:

ha.cf	General settings, heartbeat media, cluster nodes
haresources	The services that will be managed
authkeys	Encryption settings

The ha.cf file looks the same on both systems and has the entries shown in Example 9:

Example 9   heartbeat config file ha.cf

```
logfile /var/log/ha-log
logfacility     local0
keepalive 2
deadtime 10
initdead 120
serial  /dev/ttyZ0
udpport 694
udp     eth0
node    lnx31
node    lnx41
```

The values in ha.cf are the default ones, except for the serial device (see “Serial CTC” on page 36) and the nodes that make up the cluster. The serial connection is used as a non-IP heartbeat media, and in addition, udp heartbeats are send through the Ethernet (OSA) connection. The node statements represent the two systems in the cluster.

We didn’t use encryption in our tests, so the authkey file (shown in Example 10) looked the same on every system.

Example 10   heartbeat config file authkeys

```
auth 1
1 crc
#2 sha1 HI!
#3 md5 Hello!
....
```

The settings for the haresources file are described in the sections for each scenario.

In order to automatically assign the virtual IP address (cluster address) to a network interface, heartbeat needs a corresponding route to be configured for at least one interface. Because our virtual IP address was 10.1.1.1, we added the following route in /etc/route.conf (Example 11) for the eth0 device on every system:

Example 11 /etc/route.conf

```
192.168.1.0       0.0.0.0       255.255.255.0      eth0
10.1.1.0          0.0.0.0       255.255.255.0      eth0
default           192.168.1.254
.....
```

This ensures, that the virtual address alias is always assigned to the eth0 device, which is our OSA-2 connection to the LAN.

Note: In some situations where one network device fails, a deadlock situation can appear. In case of an OSA failure on one system, both sides detect that the network device on the other side isn't responding to udp pings any longer, but there is no mechanism within heartbeat which determines if one's own adapter is still working or not. This may result in a situation that the master server does not detect that his own OSA adapter may be non-operational, and thus, heartbeat is not releasing the resources/initiating the takeover. Therefore, we used mon (see "mon" on page 36) to permanently ping the outside gateway in order to verify the network connectivity to the outside world.

Serial CTC

To establish a serial connection via CTC you have to use the CTC driver with protocol-id 2. This is described in detail in the Linux for zSeries Device Driver Manual (http://www10.software.ibm.com/developerworks/opensource/linux390/documentation-2.4.7.shtml).

First you need to create the tty device within Linux:

```
# mknod /dev/ttyZ0 c 43 0
```

Add the following options for CTC in /etc/chandev.conf:

```
ctc0,0x610,0x611,0,2
```

After loading the CTC module with **modprobe** on both systems, you can test the connection. On one system do a:

```
# cat /dev/ttyZ0
```

On the other side do:

```
# echo hello > /dev/ttyZ0
```

You should see the message popping up on the first system if the connection is working correctly. Also test the connection the other way around.

Note: At the time of writing the CTC driver contained a bug in conjunction with the use of protocol-id 2. This caused a loop in the kernel process ksoftirqd_CPU, taking up to 100% CPU time. However, the system was still very responsive because we used a dual processor configuration and we could still perform our tests. The bug is reported to the developers. The first test with an internal fix showed that CPU consumption is reduced and normal. The fix has been distributed to the maintainer for further testing, and we expect it will be available soon.

mon

We utilized the monitoring tool mon to supplement the functions of heartbeat. The mon package is available from:

http://www.kernel.org/software/mon

As mentioned before, we used mon to permanently test the network connection to the gateway and thereby verify the operational status of the network adapter.

The setup of mon is described in the package documentation and will not be explained here in detail. For our purposes, 3 files need to be configured:

mon.cf The main configuration file for mon located in /etc/mon/

stop_hb An action script that gets executed and informs heartbeat in case of OSA failure

mon An init script to start mon

To monitor the network connection to the gateway via the **fping** command, the mon configuration file looks like Example 12.

Example 12 mon configuration file mon.cf

```
#
# global options
#
cfbasedir   = /usr/lib/mon/etc
alertdir    = /usr/lib/mon/alert.d
mondir      = /usr/lib/mon/mon.d
maxprocs    = 20
histlength = 100
randstart = 60s
authtype = getpwnam

#
# group definitions (hostnames or IP addresses)
#
hostgroup tirnanog 192.168.1.254

watch tirnanog
    service fping
        interval 2s
        monitor fping.monitor
            period wd {Mon-Sun}
                alertafter 3
                numalerts 1
                alertevery 1h
                alert file.alert -d /var/log ha-log
                alert stop_hb
```

For the general settings and meanings of the parameters, refer to the mon documentation. We only have one host defined, which is our gateway (tirnanog with IP address 192.168.1.254). We used **fping** to permanently check the network connection to the gateway. In case of a network interruption (for example, an OSA or cable failure), the two alert scripts file.alert and stop_hb get executed. While file.alert just logs an entry to the heartbeat logfile for later analysis, stop_hb stops heartbeat and thus initiates the takeover to the second machine. The file.alert script comes with the mon package and is shown in Example 13.

Example 13 alert script stop_hb

```
#!/bin/sh
#
EXISTS=`ps -ef | grep heartbeat | grep -v grep`
if [ -n "$EXISTS"  ]; then
/etc/init.d/heartbeat stop
fi
```

The init script we used for mon is listed in the Appendix. It can be used to automatically start mon at system boot time by placing the usual start/stop links for the runlevels in /etc/init.d/rc#.d/.

Apache

For testing purposes we used Apache version 1.3.19-40, which comes with SuSE SLES 7.

Except for changing the ServerName, DocumentRoot and Directory statements pointing to our shared filesystems, we worked with the default configuration file. As an example, the edited lines for system lnx31 with a shared r/o filesystem, mounted on /ro, are shown in Example 14.

Example 14   Changes in httpd.conf

```
ServerName lnx31.lcoc.boeblingen.de.ibm.com
...
# DocumentRoot "/usr/local/httpd/htdocs"
DocumentRoot "/ro"
...
# <Directory "/usr/local/httpd/htdocs">
<Directory "/ro">
...
# <Files /usr/local/httpd/htdocs/index.htm*>
<Files /ro/index.htm*>
```

The settings (except ServerName) are the same on both cluster nodes

Samba

For testing purposes we used Samba version 2.2.0a-21, which comes with SLES 7.

We did the necessary changes to smb.conf in order to use it as a file server for a Windows 2000 client. Refer to the Samba documentation on how to do this. The relevant changes for our environment were the binding to our virtual cluster IP address and the configuration of a share on our r/w shared DASD volume, mounted on /rw. The statements shown in were added in smb.conf.

Example 15   Changes in smb.conf

```
interfaces = 10.1.1.1/255.255.255.0
...
[rw]
   comment = Read-Write-Volume
   path = /rw
   read only = no
   public = yes
   browseable = yes
   force user = root
```

The settings are the same on both systems. The connection and read/write access to the share were tested from a Windows 2000 client on the LAN.

Zebra

The router failover test was performed with lnx31 and lnx32. In addition, a third Linux guest, lnx33 was configured as the destination for the TCP/IP requests. The system configuration for the router failover test case is shown in the Figure 16 on page 52.

To enable Linux to support dynamic routing protocol, the zebra package must be installed. Zebra is a routing software that provides TCP/IP-based routing services. The package is available either from http://www.zebra.org or from a SuSE-mirror. We downloaded the source rpm from a SuSE-mirror at:

ftp://ftp.suse.com/pub/suse/i386/7.3/full-names/src/zebra-0.92a-50.src.rpm

Zebra must be installed on all nodes. To build the zebra binary rpm you have to install the following packages:

```
#rpm -ivh readline-2.05-39.s390.rpm
#rpm -ivh readline-devel-2.05-39.s390.rpm
```

The GCC compiler is necessary for the build of the zebra-binary-rpm. If it is not already installed on your system, install the GCC as described in the GCC documentation.

The binary package is created with:

```
#rpm --rebuild zebra-0.92a-50.src.rpm
```

This creates the binary package in /usr/src/packages/RPMS. Then you can install the package with:

```
#rpm -ivh /usr/src/packages/RPMS/s390/zebra-0.92a-50.s390.rpm
```

Furthermore, you have to configure Zebra by editing two configuration files:

zebra.conf The configuration of the zebra routing manager

ospf.conf The configuration file for the ospf protocol daemon

With the exception of the hostname, the zebra.conf is the same on the two Linux systems (lnx31,lnx32). The zebra.conf file from Linux lnx31 is shown in Example 16.

Example 16 Zebra configuration file zebra.conf

```
!
! Zebra configuration for lnx31
!
hostname lnx31
password zebra
enable password zebra
log file /var/log/zebra.log
log record-priority
service advanced-vty
!
interface lo
!
interface eth0
!
interface eth1
!
interface ctc0
!
access-list term permit 127.0.0.1/32
access-list term deny any
!
line vty
 access-class term
!
```

The OSPF areas and network parameters are specified in the OSPF configuration file. In our test setup we used two OSPF areas for the subnet 10.33.0.1/32 and the 192.168.0.0/24 subnet. With the exception of the OSPF router-id, the ospf.conf is the same on the two linux systems (lnx31,lnx32). The ospf.conf file from Linux lnx31 is shown in Example 17.

Example 17 The Zebra configuration file ospf.conf

```
!
! ospfd configuration for lnx31
!
hostname lnx31
password zebra
enable password zebra
log file /var/log/ospfd.log
log record-priority
service advanced-vty
!
interface lo
!
interface eth0
!
interface eth1
!
interface ctc0
 ip ospf network point-to-point
!
router ospf
 ospf router-id 192.168.1.3
 redistribute connected route-map just-10
 redistribute connected route-map just-192
 passive-interface lo
 network 192.168.1.0/24 area 0
 network 10.33.0.1/32 area 1
!
access-list net-10 permit 10.0.0.0/8
access-list net-192 permit 192.168.1.0/24
access-list term permit 127.0.0.1/32
access-list term deny any
!
route-map just-10 permit 10
 match ip address net-10
!
route-map just-192 permit 10
 match ip address net-192
!
line vty
 access-class term
!
```

To make sure that Zebra is started at boot time, edit /etc/rc.config and add following lines:

```
START_ZEBRA=yes
START_OSPFD=yes
```

The Zebra start scripts are located in the /etc/init.d directory. To start Zebra and OSPF daemons, type:

```
# /etc/init.d/zebra start
# /etc/init.d/ospfd start
```

Zebra was also installed on lnx33 and the router tirnanog. The configuration files zebra.conf and ospf.conf differ from the files on lnx31 and lnx32. Linux systems lnx33 and tirnanog don't distribute messages; they only listen on messages to detect any changes and to update the routing table. The zebra.conf file for router tirnanog is given in Example 18.

Example 18 Zebra configuration file on router tirnanog zebra.conf

```
tirnanog/zebra.conf
!
! Zebra configuration for tirnanog
!
hostname tirnanog
password zebra
enable password zebra
log file /var/log/zebra.log
log record-priority
service advanced-vty
!
!interface lo
!
interface eth0
!
! to limit the access of the configuration-console to localhost:
access-list term permit 127.0.0.1/32
access-list term deny any
!
line vty
 access-class term
```

The ospf.config for tirnanog is given in Example 19.

Example 19 Zebra configuration file ospf.conf on router tirnanog

```
tirnanog/ospfd.conf
!
! ospfd configuration for tirnanog
!
hostname tirnanog
password zebra
enable password zebra
log file /var/log/ospfd.log
log record-priority
service advanced-vty
!
!
interface lo
!
interface eth0
!
router ospf
 ospf router-id 192.168.1.254
 passive-interface lo
 network 192.168.1.0/24 area 0
!
! to limit the access of the configuration-console to localhost:
access-list term permit 127.0.0.1/32
access-list term deny any
!
```

```
line vty
 access-class term
!
```

The zebra.conf and the ospf.conf files for destination Linux lnx33 are presented in the Appendix.

High availability test scenarios

The following tests were performed during this project:

- OSA adapter failure
- Apache WebServer failover
- Samba server failover
- Linux router failover

OSA adapter failure

Description

The first scenario we tested was the OSA adapter failure with one server; the configuration for this test is shown in Figure 11 on page 44. The Linux guest system lnx31 was configured with two Ethernet interfaces that are directly connected over the OSA adapters to the router tirnanog. The IP addresses of the Linux system are in the same subnet. The virtual IP address necessary for the adapter takeover was assigned to the dummy interface, in our case the loopback interface (lo:0).

Implementation

To assign the IP address 192.168.1.3 to Ethernet interface eth0 the following **ifconfig** command was issued:

```
lnx31# ifconfig eth0 192.168.1.3 netmask 255.255.255.0
broadcast 192.168.1.255
```

The second Ethernet interface was assigned IP address 192.168.1.4:

```
lnx31# ifconfig eth1 192.168.1.4 netmask 255.255.255.0
broadcast 192.168.1.255
```

The VIPA address on the loopback interface was configured with:

```
lnx31# ifconfig lo:0 10.1.1.1
```

To get the hardware address of the interfaces we issued:

```
lnx31# ifconfig
```

The output is shown in Example 20. The eth0 interface has hardware address 00:20:35:04:DE:CF and eth1 00:20:35:04:9E:B2.

Example 20 Output of ifconfig after the interface configuration

```
eth0      Link encap:Ethernet  HWaddr 00:20:35:04:DE:CF
          inet addr:192.168.1.3  Bcast:192.168.1.255  Mask:255.255.255.0
          inet6 addr: fe80::220:35ff:fe04:decf/10 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:221 errors:0 dropped:0 overruns:0 frame:0
          TX packets:111 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:100
          RX bytes:36188 (35.3 Kb)  TX bytes:17185 (16.7 Kb)

eth1      Link encap:Ethernet  HWaddr 00:20:35:04:9E:B2
          inet addr:192.168.1.4  Bcast:192.168.1.255  Mask:255.255.255.0
          inet6 addr: fe80::220:35ff:fe04:9eb2/10 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:18860 errors:0 dropped:0 overruns:0 frame:0
          TX packets:11687 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:100
          RX bytes:2234568 (2.1 Mb)  TX bytes:1559406 (1.4 Mb)

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          inet6 addr: ::1/128 Scope:Host
          UP LOOPBACK RUNNING  MTU:16436  Metric:1
          RX packets:233 errors:0 dropped:0 overruns:0 frame:0
          TX packets:233 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:40262 (39.3 Kb)  TX bytes:40262 (39.3 Kb)

lo:0      Link encap:Local Loopback
          inet addr:10.1.1.1  Mask:255.0.0.0
          UP LOOPBACK RUNNING  MTU:16436  Metric:1
```

The Ethernet interface of the router tirnanog was configured by the following command:

```
# ifconfig eth0 192.168.1.254 netmask 255.255.255.0
broadcast 192.168.1.255
```

A static network route to subnet 10.1.1.0/24 at the Ethernet interface of the router tirnanog was added:

```
# route add -net 10.1.1.0 netmask 255.255.255.0 dev eth0
```

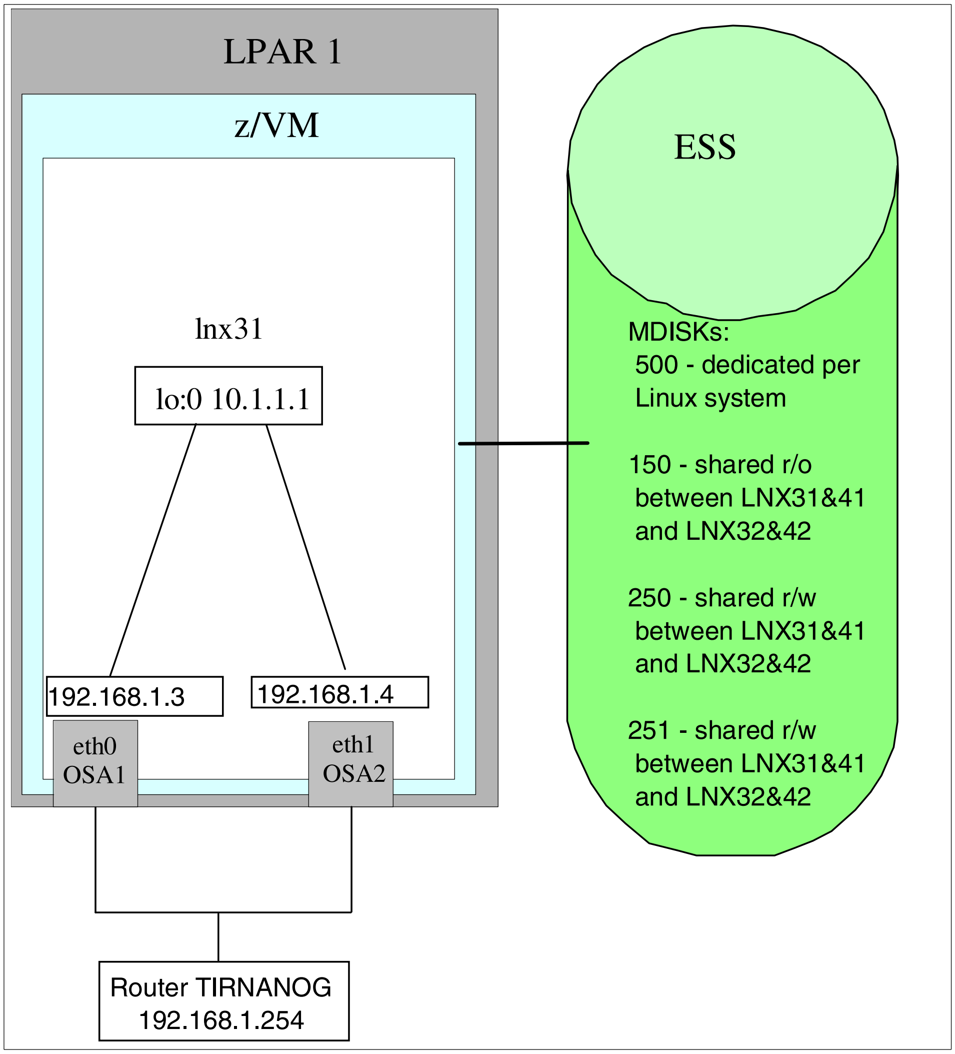

Figure 11 Configuration for OSA adapter failure

In this test we don't use the OSPF protocol, therefore we stopped the zebra daemons:

```
lnx31# /etc/init.d/zebra stop
lnx31# /etc/init.d/ospf stop
```

Test

To test the failure of one adapter we detached the virtual device from the Linux (lnx31) guest by issuing the following command on the z/VM console:

```
#cp det a02-a03
```

For the Linux system it seems that either a network adapter failure or cable failure occurred.

To show that an adapter takeover took place we pinged the virtual address and observed whether a failure occurred. In addition, we checked the ARP table of the router tirnanog to see if the route to the virtual machine has been changed.

Before detaching the OSA adapter, we had the ARP table presented in Example 21 on page 45. The ARP table shows that the requests were routed over Ethernet interface eth0.

Example 21   ARP cache for router tirnanog before OSA failure

```
Address                 HWtype  HWaddress           Flags Mask          Iface
10.1.1.1                ether   00:20:35:04:DE:CF   C                   eth0
192.168.1.3             ether   00:20:35:04:DE:CF   C                   eth0
192.168.1.4             ether   00:20:35:04:9E:B2   C                   eth0
```

After detaching the OSA adapter, the output for the ARP cache shown in Example 22 was obtained. As you can see, the request used the eth1 interface of lnx31(see hardware address in Example 22).

Example 22   ARP cache of router tirnanog after OSA failure

```
Address                 HWtype  HWaddress           Flags Mask          Iface
10.1.1.1                ether   00:20:35:04:9E:B2   C                   eth0
192.168.1.3             ether   00:20:35:04:DE:CF   C                   eth0
192.168.1.4             ether   00:20:35:04:9E:B2   C                   eth0
```

A limitation of this scenario is that we have to wait 50 seconds until the ARP cache of the router is refreshed. During this time the router has no route to the VIPA. The other limitation of this configuration is that lnx31 itself doesn't detect the OSA failure and no route update is performed. This means that a ping from lnx31 to tirnanog failed.

High availability Apache Web servers

Description

This scenario consists of an IP and an application takeover, assuming that the data are shared read-only between both environments. This means that no data takeover is included in this scenario. Since the application chosen was the Apache Web server, the configuration data are located on a read-only shared DASD. The goal is to provide continuous availability (or minimum disruption) for the end user who is connecting to the HA cluster Web server via a Web browser. In case of any failure (either the Linux system, the network, or the machine itself), heartbeat or mon will detect the outage and initiate the takeover to move the resources to the secondary machine.

Implementation

In our example, when in default mode, the Apache Web server runs on system lnx31 while the secondary system lnx41 is active without a running Apache Web server.

Figure 12 on page 46 shows the configuration we used. This is the standard operational mode.

The haresources file on both systems contains only the following line:

```
lnx31 10.1.1.1 apache
```

This sets the system lnx31 as the primary system, 10.1.1.1 as the virtual IP address for the cluster, and handles the apache init script for starting and stopping the Web server. Refer to the heartbeat documentation for a more detailed explanation of the syntax for the haresources file. The Web server is managed by the heartbeat daemon and listens on virtual address 10.1.1.1, which is an alias to the network interface eth0 (OSA adapter).

The Apache Web server is configured so that the content data reside on the shared r/o volume /dev/dasdb which is concurrently mounted on each side. Thus no data or file system takeover is needed.

The heartbeat daemon and mon daemon have to be started on each system.

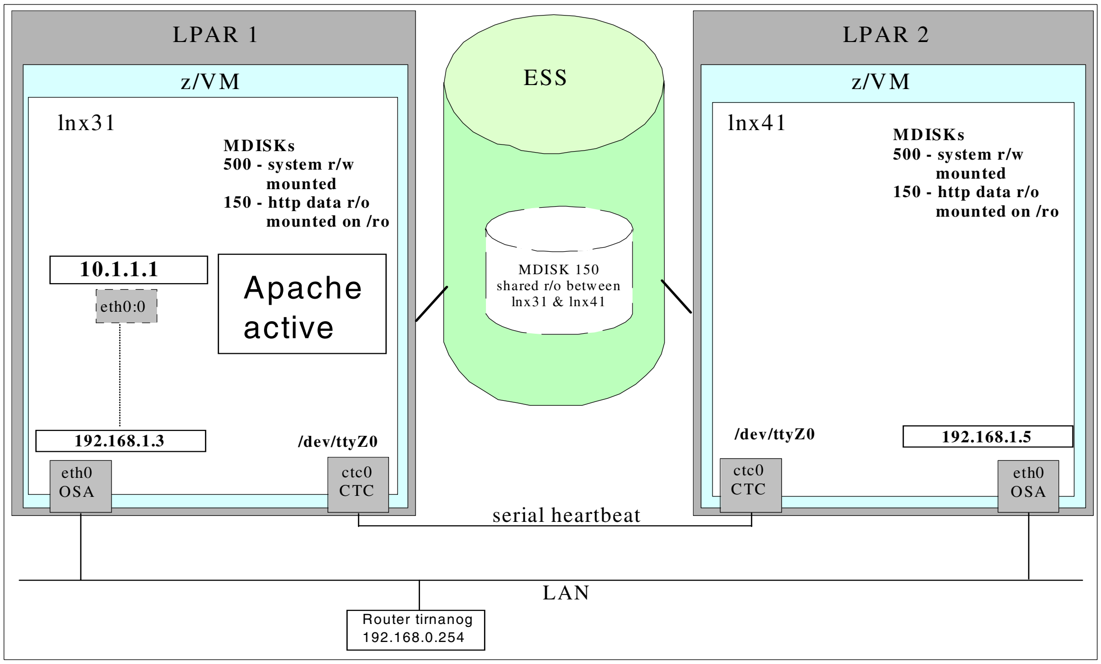

Figure 12   Test setup Apache failover

When the client connects to IP address 10.1.1.1, the request actually gets answered by the system lnx31, where Apache is running in normal mode.

Test

To test the scenario, we simulated two outages on system lnx31 (see Figure 13):

1. A complete machine outage
2. Network/OSA failure

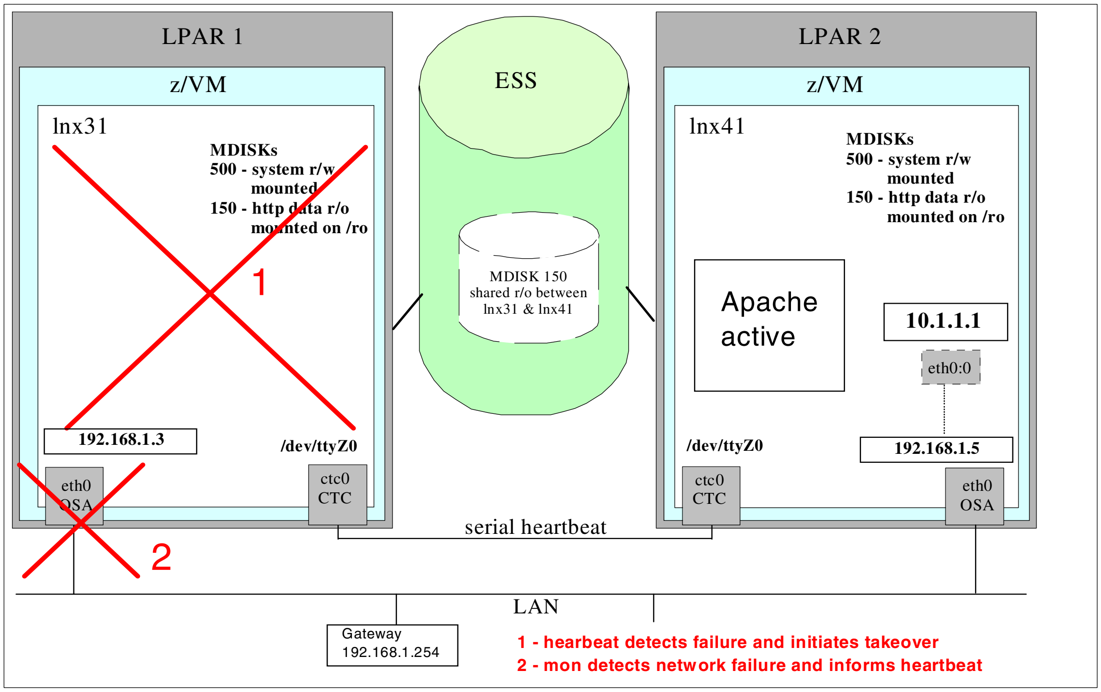

Figure 13   Apache Failover Test Scenarios

The first outage was simulated by issuing the following CP command on the VM console:

```
#cp sleep 5min
```

This stops the guest machine lnx31 for 5 minutes, so the heartbeat process on lnx41 detects that system lnx31 is not responding to any heartbeats any longer. This is equivalent to a machine that crashed completely for whatever reason.

Then the heartbeat process on lnx41:

- Acquires the resources, which in this case means it activates IP address 10.1.1.1 as an alias on its eth0 interface
- Sends out a gratuitous ARP to inform the other systems/routers on the LAN that the IP address 10.1.1.1 is now reachable on system lnx41
- Starts the Apache Web server to listen on that IP address

The log file for this example is taken from system lnx41 and is shown in Example 23.

Example 23   heartbeat log file for Apache takeover

```
heartbeat: 2002/05/15_12:35:40 info: **************************
heartbeat: 2002/05/15_12:35:40 info: Configuration validated. Starting heartbeat 0.4.9.1
heartbeat: 2002/05/15_12:35:40 info: heartbeat: version 0.4.9.1
heartbeat: 2002/05/15_12:35:40 info: Heartbeat generation: 4
heartbeat: 2002/05/15_12:35:40 info: Creating FIFO /var/run/heartbeat-fifo.
heartbeat: 2002/05/15_12:35:40 notice: Starting serial heartbeat on tty /dev/ttyZ0
heartbeat: 2002/05/15_12:35:40 notice: UDP heartbeat started on port 694 interface eth0
heartbeat: 2002/05/15_12:35:40 info: Local status now set to: 'up'
heartbeat: 2002/05/15_12:35:42 info: Heartbeat restart on node lnx41
heartbeat: 2002/05/15_12:35:42 info: Link lnx41:eth0 up.
heartbeat: 2002/05/15_12:35:42 info: Local status now set to: 'active'
heartbeat: 2002/05/15_12:35:42 info: Heartbeat restart on node lnx31
heartbeat: 2002/05/15_12:35:42 info: Link lnx31:eth0 up.
heartbeat: 2002/05/15_12:35:42 info: Node lnx31: status up
heartbeat: 2002/05/15_12:35:42 info: Node lnx31: status active
heartbeat: 2002/05/15_12:35:42 info: Node lnx41: status up
heartbeat: 2002/05/15_12:35:42 info: Node lnx41: status active
heartbeat: 2002/05/15_12:35:42 info: Running /etc/ha.d/rc.d/status status
heartbeat: 2002/05/15_12:35:42 info: Running /etc/ha.d/rc.d/ifstat ifstat
heartbeat: 2002/05/15_12:35:42 info: Running /etc/ha.d/rc.d/status status
heartbeat: 2002/05/15_12:35:42 info: Running /etc/ha.d/rc.d/status status
heartbeat: 2002/05/15_12:35:42 info: Running /etc/ha.d/rc.d/ifstat ifstat
heartbeat: 2002/05/15_12:35:42 info: Running /etc/ha.d/rc.d/status status
heartbeat: 2002/05/15_12:35:42 info: Running /etc/ha.d/rc.d/ip-request ip-request
heartbeat: 2002/05/15_12:35:42 info: No local resources [/usr/lib/heartbeat/ResourceManager listkeys lnx41]
heartbeat: 2002/05/15_12:35:42 info: Resource acquisition completed.
heartbeat: 2002/05/15_12:35:42 info: Running /etc/ha.d/resource.d/IPaddr 10.1.1.1 status
heartbeat: 2002/05/15_12:35:45 info: Link lnx31:/dev/ttyZ0 up.
heartbeat: 2002/05/15_12:35:45 info: Running /etc/ha.d/rc.d/ifstat ifstat
heartbeat: 2002/05/15_12:38:17 info: Link lnx31:eth0 dead.
heartbeat: 2002/05/15_12:45:29 info: Running /etc/ha.d/rc.d/ifstat ifstat
heartbeat: 2002/05/15_12:46:26 info: Link lnx31:/dev/ttyZ0 dead.
heartbeat: 2002/05/15_12:46:26 WARN: node lnx31: is dead
heartbeat: 2002/05/15_12:46:26 info: Link lnx31:eth0 dead.
heartbeat: 2002/05/15_12:46:26 info: Running /etc/ha.d/rc.d/status status
heartbeat: 2002/05/15_12:46:26 info: Running /etc/ha.d/rc.d/ifstat ifstat
heartbeat: 2002/05/15_12:46:26 info: Running /etc/ha.d/rc.d/ifstat ifstat
heartbeat: 2002/05/15_12:46:26 info: Taking over resource group 10.1.1.1
heartbeat: 2002/05/15_12:46:26 info: Acquiring resource group: lnx31 10.1.1.1 apache
heartbeat: 2002/05/15_12:46:26 info: Running /etc/ha.d/resource.d/IPaddr 10.1.1.1 start
heartbeat: 2002/05/15_12:46:26 info: ifconfig eth0:0 10.1.1.1 netmask 255.255.255.0   broadcast 10.1.1.255
```

```
heartbeat: 2002/05/15_12:46:26 info: Sending Gratuitous Arp for 10.1.1.1 on eth0:0 [eth0]
heartbeat: 2002/05/15_12:46:26 info: Running /etc/rc.d/init.d/apache  start
heartbeat: 2002/05/15_12:46:29 info: mach_down takeover complete.
```

The complete takeover process was done within 2 second in this scenario. After the system lnx31 was reactivated, the heartbeat process on system lnx41 released the resources and the Web server was automatically restarted from heartbeat with the default Apache start script on system lnx31.

The second outage was simulated by issuing the following command on the z/VM console:

```
#cp det a02
```

This detaches the read channel of the OSA adapter from the guest machine lnx31. This is transparent to the Linux operating system and therefore nothing changes within the interface configuration or routing information on system lnx31. However, the network is no longer reachable, which is equivalent to a network adapter or cable failure!

Both heartbeat processes on each system detect that the other node is no longer responding to udp pings via Ethernet, and therefore declare the other eth0 link as dead. Since heartbeat itself has no mechanism to detect if the interface on its own system or on the other side is dead, our mon process comes into play. Mon detects that the gateway is no longer reachable from system lnx31, and therefore shuts down the heartbeat process on system lnx31. This automatically releases the resources on system lnx31 and—since the systems still communicate over the serial connection—restarts the virtual IP address and the Web server on system lnx41 the same way as in test 1.

You see the following messages (Example 24) on the z/VM console:

Example 24   mon messages on z/VM console

```
May 22 12:55:43 lnx31 mon¬3506|: failure for tirnanog fping 1022063743 192.168.1.254
May 22 12:55:46 lnx31 mon¬3506|: failure for tirnanog fping 1022063746 192.168.1.254
May 22 12:55:49 lnx31 mon¬3506|: failure for tirnanog fping 1022063749 192.168.1.254

May 22 12:55:49 lnx31 mon¬3506|: calling alert file.alert for tirnanog/fping
(/usr/lib/mon/alert.d/file.alert,-d /var/log ha-log) 192.168.1.254
May 22 12:55:49 lnx31 mon¬3506|: calling alert stop_hb for tirnanog/fping
(/usr/lib/mon/alert.d/stop_hb,) 192.168.1.254
```

An additional entry in the heartbeat log file on lnx31 (Example 25) is created; it can be used in later analysis to determine the cause of the takeover.

Example 25   mon entry in heartbeat log file

```
failure tirnanog fping 1022068058 (Wed May 22 13:47:38) 192.168.1.254
DESCR=
OPSTATUS=0

start time: Wed May 22 13:47:36 2002
end time  : Wed May 22 13:47:38 2002
duration  : 2 seconds

-----------------------------------------------------------------------------
unreachable hosts
-----------------------------------------------------------------------------
192.168.1.254
```

The second takeover, due to the OSA/network failure, only took a little longer than the first one.

During the tests, we verified the connection to the Web server by continuously reloading a Web page from an additional client. Each Web server served out simple static pages which, among other things, contained the real name of the system. There was almost no disruption on the client side; the only difference in behavior detectable was the changing system name.

High availability Samba server

Description

In this scenario (shown in Figure 14), in addition to the application and IP takeover, data takeover is included. The data takeover means that the disks on which the data resides must be shared in a read/write mode. A shared Linux file system that allows read/write access was not used because we want to show the failover with standard Linux tools. z/VM supports the assignment of a DASD to more than one Linux guest system. During takeover the file system must be unmounted from the failing system and then mounted on the secondary system. The LVM is often used to increase the I/O performance. Thus we included the LVM takeover to see whether the LVM can participate on a high availability solution. SMB (Samba) is used to serve data for Windows clients in the scope of this HA scenario.

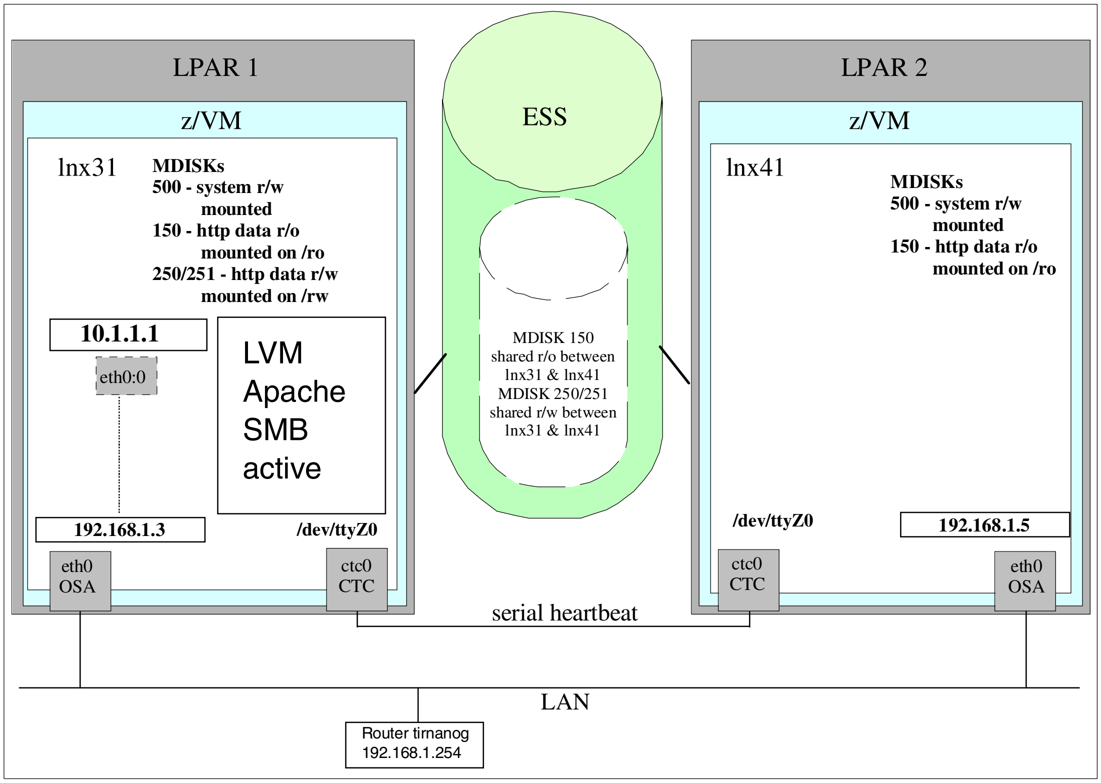

Figure 14 Setup for SAMBA server failover

Implementation

Both Linux guests are configured to have access to the r/w shared DASD 250 and DASD 251, but only the Linux server lnx31 (active) has mounted the file system on the z/VM shared disk (see Figure 14). The haresources file has to be changed because we have new resources that must be started and stopped during the takeover.

The haresoures file contains the following line on both sides:

```
lnx31 10.1.1.1 lvm Filesystem::/dev/vg00/rw::/rw::reiserfs apache smb
```

This sets lnx31 as the primary node, with virtual IP address 10.1.1.1. The start order of the resources that heartbeat manages is the order the resources appear in this line. This means the following start order is used in our test:

1. LVM
2. Mount the filesystem
3. Apache
4. SMB (Samba)

The heartbeat daemon and mon have to be started on each system. Figure 14 shows the configuration we used.

Test

To test the scenario we simulated two outages on system lnx31 (Figure 15):

1. A complete machine outage
2. Network/OSA failure

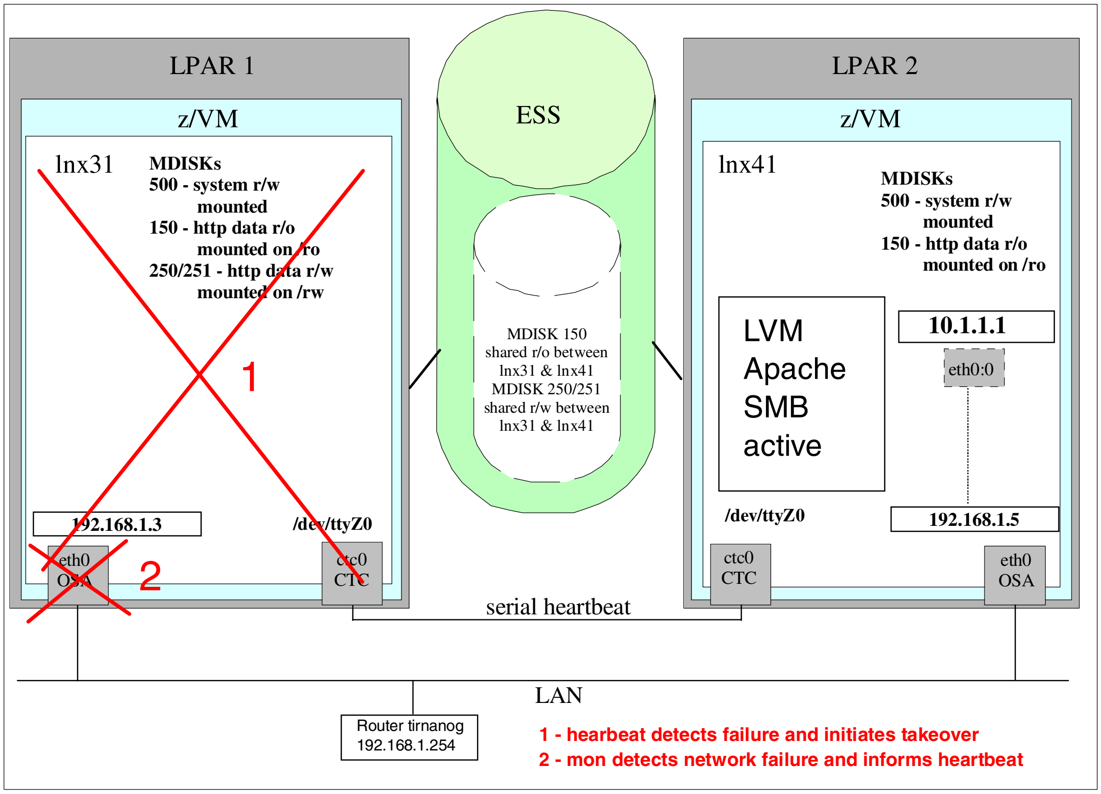
LPAR 1
z/VM
lnx31
MDISKs
500 - system r/w mounted
150 - http data r/o mounted on /ro
250/251 - http data r/w mounted on /rw
1
192.168.1.3
/dev/ttyZ0
eth0 OSA
2
ctc0 CTC
ESS
MDISK 150 shared r/o between lnx31 & lnx41
MDISK 250/251 shared r/w between lnx31 & lnx41
LPAR 2
z/VM
lnx41
MDISKs
500 - system r/w mounted
150 - http data r/o mounted on /ro
LVM
Apache
SMB
active
10.1.1.1
eth0:0
/dev/ttyZ0
192.168.1.5
ctc0 CTC
eth0 OSA
serial heartbeat
LAN
Router tirnanog 192.168.1.254
1 - hearbeat detects failure and initiates takeover
2 - mon detects network failure and informs heartbeat

Figure 15 Failover Samba test cases

1. To simulate the server failure, we issued the **cp force** command on the z/VM console which stops the guest:

   ```
   #cp force lnx31
   ```

   Note: The test with the **cp sleep** command is a special case of a partitioned cluster. The Linux guest was stopped immediately without releasing the resources. After the wait time, the guest becomes active and for a short time, until the second Linux server detects the availability of the primary server, the shared resource is accessible for both servers. This may result in damaged files (see more details in “Stonith” on page 26).

Heartbeat on lnx41 detects the failure of lnx31 and responds with a takeover of the resources. The output of the heartbeat log file is almost identical with the log file from the Apache failover test (see Example 23 on page 47). Thus only the differences to the Apache test log file are shown here(Example 26).

Example 26   Heartbeat Logfile Output lnx41

```
heartbeat: 2002/05/24_16:14:01 info: Taking over resource group 10.1.1.1

heartbeat: 2002/05/24_16:14:01 info: Acquiring resource group: lnx31 10.1.1.1 lvm
Filesystem::/dev/vg00/rw::/rw::reiserfs apache smb

heartbeat: 2002/05/24_16:14:01 info: Running /etc/ha.d/resource.d/IPaddr 10.1.1.1 start

heartbeat: 2002/05/24_16:14:01 info: ifconfig eth0:0 10.1.1.1 netmask 255.255.255.0   broadcast 10.1.1.255

heartbeat: 2002/05/24_16:14:01 info: Sending Gratuitous Arp for 10.1.1.1 on eth0:0 [eth0]

heartbeat: 2002/05/24_16:14:01 info: Running /etc/rc.d/init.d/lvm  start

heartbeat: 2002/05/24_16:14:02 info: Running /etc/ha.d/resource.d/Filesystem /dev/vg00/rw /rw reiserfs
start

heartbeat: 2002/05/24_16:14:02 WARNING: Couldn't insert reiserfs module. (usually safe to ingore)

heartbeat: 2002/05/24_16:14:02 info: Running /etc/rc.d/init.d/apache  start

heartbeat: 2002/05/24_16:14:04 info: Running /etc/rc.d/init.d/smb  start

heartbeat: 2002/05/24_16:14:05 info: mach_down takeover complete.
```

We see that heartbeat detected the failure and acquired the resources. First the virtual IP address was activated and a gratuitous ARP was sent. After the LVM start, the file system was mounted. The warning in the log file can be ignored. At the end, Apache and smb are activated.

Before the failure was simulated, we mapped the Samba share from the Windows client and checked that we could access the file system. We didn’t disconnect from the Samba share. After failover we opened the file successfully again. We didn’t reconnect to the share, thus the failover of the Linux server was transparent to the client.

Note: In our test, no program was running that uses a file. If a program runs that needs resources on the share, it may happen that the program gets an error. In this case the program must be able to handle the error and initiate a retry of the operation.

2. The network adapter failure test was also successful. The messages seen on the console are the same as in the Apache failover test (see Examples 23, 24,and 25) and therefore they are not shown again here.

High availability Linux router

Description

The router of a virtual LAN is a single point of failure. To ensure high availability of the router we set up two Linux guests as routers. A third Linux guest (lnx33) was configured with a CTC point-to-point connection to both routers to simulate a host in the virtual LAN (see Figure 16 on page 52). A failure of one router must be advertised to the other routers in the net. With dynamic routing protocols, changes due to failure of routes in the network are handled automatically. In our test we used the Open Shortest Path First (OSPF) protocol. Each router broadcasts link status information to each of its neighboring routers. Each router uses the status information to build a complete routing table. OSPF supports the concept of "areas," where the autonomous system is divided into areas, each responsible for its own topology. Area topology is not propagated to other areas. Border routers maintain the connectivity between the separate areas.

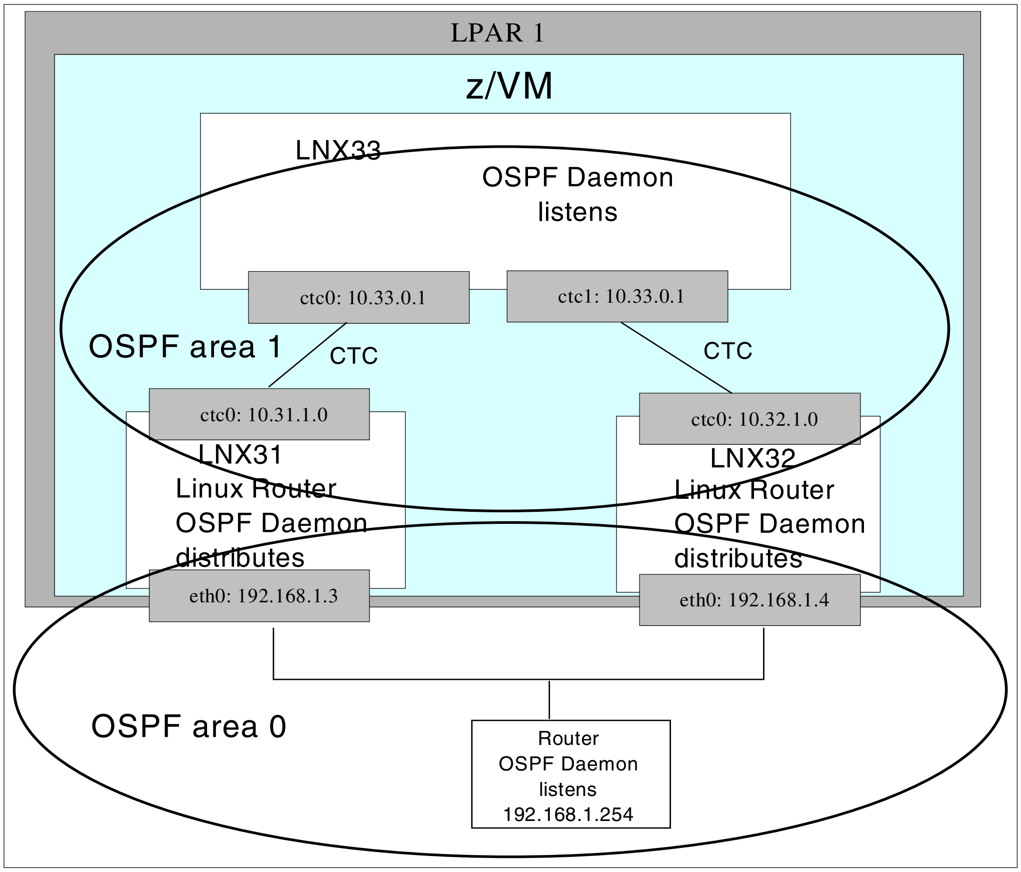

Figure 16 Configuration of Linux router failover test

In our test we used two areas (areas 0,1 in Figure 16). The area inside z/VM (subnet 10.33.0.1/32) and the subnet 192.168.1.0/24 behind z/VM.

Implementation

The configuration of the OSPF daemons is described in the section "Software setup" on page 32. Linux lnx31 and lnx32 are configured to send messages to tirnanog and lnx33 (see redistribute statement in OSPF configuration file in Example 27).

Example 27 OSPF statements in ospf.conf file for Linux server lnx32

```
ospf router-id 192.168.1.4
redistribute connected route-map just-10
redistribute connected route-map just-192
passive-interface lo
 network 192.168.1.0/24 area 0
```

The statements shown in Example 27 are not contained in the ospf.conf of the Linux systems lnx33 and tirnanog because they don't distribute OSPF messages.

Since lnx31 and lnx32 are border routers, they have two OSPF areas defined, as shown in Example 28.

Example 28   OSPF network area definitions in ospf.conf file for Linux servers lnx31 and lnx32

```
network 192.168.1.0/24 area 0
network 10.33.0.1/32 area 1
```

Linux systems lnx33 and tirnanog belong to one OSPF area, and only one area is defined in each Linux server ospf.conf file. The definition for Linux lnx33 is shown in Example 29.

Example 29   OSPF network area definitions in ospf.conf file for Linux server lnx33

```
network 10.31.0.1/32 area 1
network 10.32.0.1/32 area 1
```

Test

The failure of a router was simulated with the **cp sleep** command on the z/VM console of Linux guest lnx31.

```
#cp sleep 5 min
```

To check that the OSPF daemons work correctly we compared the routing table of lnx33 before and after the failure.

With the **route** command we displayed the routing table on the screen:

```
lnx33# route -n
```

The output of the routing table before the test start is shown in Example 30. The destination IP address of Linux tirnanog is reached over gateway 10.31.0.1 that is router lnx31.

Example 30   Routing table before OSA failure (Linux lnx33)

```
Destination     Gateway         Genmask          Flags Metric  Ref    Use Iface
 10.33.0.1       10.31.0.1       255.255.255.255  UGH   19      0       0 ctc1
 10.32.0.1       0.0.0.0         255.255.255.255  UH    0       0       0 ctc1
 10.31.0.1       0.0.0.0         255.255.255.255  UH    0       0       0 ctc0
 192.168.1.0     10.31.0.1       255.255.255.0    UG    19      0       0 ctc0
```

The corresponding routing table of Linux router tirnanog is presented in Example 31. The lnx33 is reached over gateway lnx31.

Example 31   Routing table before OSA failure (Linux tirnanog)

```
Destination     Gateway          Genmask         Flags Metric Ref    Use Iface
 10.33.0.1       192.168.1.3      255.255.255.255 UGH   d0     0       0 eth0
 10.32.0.1       192.168.1.3      255.255.255.255 UGH   30     0       0 eth0
 10.31.0.1       192.168.1.3      255.255.255.255 UGH   30     0       0 eth0
 192.168.1.0     0.0.0.0          255.255.255.0   U     0      0       0 eth0
 127.0.0.0       0.0.0.0          255.0.0.0       U     0      0       0  lo
```

After the failure the routing tables on linux lnx33 (Example 32) and tirnanog (Example 33) show that the gateway is switched from lnx31 to lnx32.

Example 32 Routing Table after OSA failure (Linux lnx33)

```
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
 10.33.0.1       10.32.0.1       255.255.255.255 UGH   19     0        0 ctc1
 10.32.0.1       0.0.0.0         255.255.255.255 UH    0      0        0 ctc1
 10.31.0.1       0.0.0.0         255.255.255.255 UH    0      0        0 ctc0
 192.168.1.0     10.32.0.1       255.255.255.0   UG    19     0        0 ctc1
```

Example 33 Routing Table after OSA failure (Linux tirnanog)

```
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
 10.33.0.1       192.168.1.4     255.255.255.255 UGH   20     0        0 eth0
 10.32.0.1       192.168.1.4     255.255.255.255 UGH   30     0        0 eth0
 10.31.0.1       192.168.1.4     255.255.255.255 UGH   30     0        0 eth0
 192.168.0.0     0.0.0.0         255.255.255.0   U     0      0        0 eth0
 127.0.0.0       0.0.0.0         255.0.0.0       U     0      0        0 lo
```

Example 34 presents the output of the OSPF log file on lnx32. It shows that the OSPF daemon detects the failure and the backup route is propagated.

Example 34 Output of OSPF daemon on lnx32

```
2002/05/22 20:16:53 informational: OSPF: nsm_change_status(): scheduling new router-LSA
origination
    2002/05/22 20:16:53 informational: OSPF: DR-Election[1st]: Backup 192.168.1.4
    2002/05/22 20:16:53 informational: OSPF: DR-Election[1st]: DR     192.168.1.254
    2002/05/22 20:16:56 warnings: OSPF: Link State Update: LS age is equal to MaxAge.
    2002/05/22 20:16:56 warnings: OSPF: Link State Update: LS age is equal to MaxAge.
```

The test presented here used a point-to-point virtual network.

Note: The configuration of a point-to-point virtual LAN is more complex. The concept of the new guest LAN feature in z/VM 4.3 makes the configuration of the network connections easier.

With z/VM version 4.2 the virtual LAN (guest LAN) is introduced. The OSPF protocol uses broadcasts to inform neighbor hosts. This feature is not supported in the guest LAN of z/VM 4.2. Thus OSPF can't be used with z/VM 4.2. Broadcast in a guest LAN is supported with z/VM 4.3 and the shown scenario can be implemented.

Customer scenario

At a German bank we recently helped the customer to implement a high availability solution for a new Web-based application on Linux for zSeries.

The challenge

At this customer we have two divisions that are able to deploy new applications: the S/390 Division and the Server Division. The latter operates all platforms other than S/390 and zSeries. Once an application project nears completion, these two divisions are approached and have to make a proposal on how they can deploy the application, both from a technical and a financial standpoint.

When this new project asked for a deployment proposal, one of the major technical issues was high-availability—at least to the extent that is common on the distributed platforms where this customer utilizes software such as Network Dispatcher to achieve this goal.

The solution

When considering how to deploy this application, the S/390 division took the following issues into account.

WebSphere Application Server for Linux on zSeries is already being used for seven productive WebSphere applications. On the other hand, WebSphere Application Server for z/OS at the moment is only being used in the old version 3.5, and no plan exists to deploy any further projects on this platform, possibly not until WebSphere Application Server Version 5 is available. So Linux was the natural choice in this particular case.

The application is based on servlets only, and there is no need for EJB support. This helped because on Linux for zSeries the currently used version of WAS is 3.5.6 and the corresponding skill within the division is sufficiently high to guarantee smooth deployment. However, the application has the need to access both DB2 on z/OS and CICS on z/OS.

Regarding the request for high availability, the customer took the stance: Let us use what we have and only implement those components which are absolutely necessary. The components available and proven were: DB2 on z/OS with data sharing in a Parallel Sysplex, CICS TS on z/OS in a CICSPLEX, and multiple Network Dispatcher configurations on the networking side.

What remained to be set up was a solution with a duplicated Linux and WebSphere Application Server. Initially a proof of concept was done under z/VM. When this was successful and convinced the project owners, the two Linux systems were implemented in separate LPARs on separate zSeries machines. It is with this configuration that the customer went into production in mid-May 2002. Presently it is an Intranet-only solution, but it might be expanded to include customers at a later stage. Figure 17 illustrates the infrastructure.

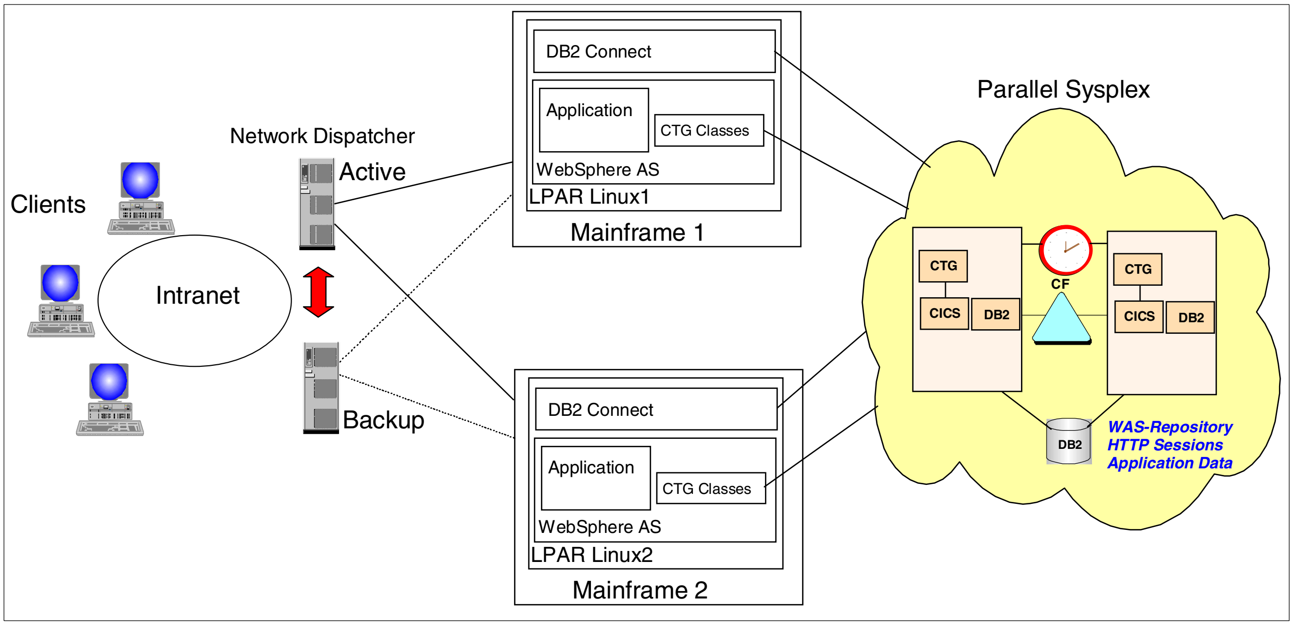

Figure 17 Infrastructure customer scenario

The figure clearly shows how existing high availability components were used to achieve the following goals.

Network high availability

Network high availability is achieved by using the Network Dispatcher software. It is a component of IBM WebSphere Edge Server and uses IP-level load balancing techniques to distribute the incoming requests to the Linux applications. The Dispatcher already provides high availability for the servers clustered behind it, because one of its basic functions is to avoid choosing a failed server. However, the Dispatcher can also be configured to eliminate the load balancer itself as a single point of failure, as part of a comprehensive hardware and software implementation of high availability. It can be configured with a secondary/standby machine on the same subnet, which listens for a heartbeat from the primary/active machine and synchronizes its state, including that of its connection table, with that of the primary/active machine. If the standby machine detects that the heartbeat from the active machine is no longer being received, it becomes active. The standby machine takes over the cluster IP addresses that are being served by the site. In this customer's case, Network Dispatcher Version 3.6 on two AIX machines is being used.

It is beyond the scope of this paper to discuss the Network Dispatcher functionality and implementation in detail. You are therefore referred to the following documentation:

Network Dispatcher, V3.6: Scalability, Availability and Load-balancing for TCP/IP Applications, available from:
http://www-3.ibm.com/software/webservers/edgeserver/library.html

WebSphere Edge Server: Working with Web Traffic Express and Network Dispatcher, SG24-6172.

Hardware high availability

Hardware high availability is achieved by running two Linux LPARs on two different zSeries machines. In fact, these two machines are in two different locations, thereby further adding to high availability. When either system fails, application requests will be routed to the remaining system automatically by Network Dispatcher until the failed system is up and running again.

System software high availability

System software high availability is achieved by having two separate Linux systems with no shared data. There is no need to share data as far as Linux itself is concerned in this configuration. The shared data all resides on DB2 on z/OS in a Parallel Sysplex. When either Linux system fails, the application requests will be routed to the remaining system automatically by Network Dispatcher until the failed system is up and running again.

Application high availability

Application high availability is achieved by deploying the same application on both Linux systems with WebSphere Application Server. Each of the two WebSphere Application Server instances has its own data, which is kept separate from the other's. This dual WebSphere Application Server layout is also called a "WebSphere Domain," and during administration one has two nodes. When either server or the application fails, the application requests will be routed to the remaining system automatically by Network Dispatcher until the failed system is up and running again.

Data high availability

Data high availability is achieved by placing all active data required by WAS and the application on databases managed within a z/OS Parallel Sysplex. This concerns the:

- Application data which is accessed via DB2 and/or CICS
- One WebSphere repository database for both instances
- HTTP Session data, which is used by WebSphere for session persistence

The project owners were convinced that the described infrastructure attains a level of high availability which cannot be reached by other platforms. In particular, the availability of the data on z/OS in the Parallel Sysplex is something no other platform can offer. Having shown the viability of such a setup, the application was taken into production.

The steps necessary to implement a remote WebSphere repository database and a remote HTTP session database are outlined in:

http://www-3.ibm.com/software/webservers/appserv/doc/v35/ae/infocenter/was/030503.html
http://www-3.ibm.com/software/webservers/appserv/doc/v35/ae/infocenter/was/030502.html

Software versions used

- Suse V7 Kernel 2.2.19 for Linux on z/Series
- WebSphere Application Server AE V3.5 Fixpak 6 for Linux on zSeries
- SDK 1.2.2
- DB2 Connect V7.1 Fixpak 3 and Fixpak 6 for Linux on zSeries
- DB2 V7 for z/OS

Future Work

While the solution in production today guarantees a very high level of availability, there are, of course, some issues still to be looked at. Among the points that require further analysis are:

- Due to limitations in the CICS Transaction Gateway (CTG) and the application design, the customer is looking into a solution where all CTG access is done by a Java bean that is able to address more than one destination and is also able to route traffic intelligently should one of the destinations fail.
- When one of the WebSphere Application Server instances takes over the work because the other one failed, it cannot presently use the session data of the failed instance. This topic is being addressed with IBM.

Summary

The objective of this paper was to show the techniques necessary to implement a high availability solution for Linux on zSeries running under z/VM. Today several high availability products for Linux exist, but non of them cover both z/VM and Linux aspects.

We showed through our testing that the key technologies of a high availability solution are available. But no out of the box solution exists. In all cases you have to consider additional issues, and create solutions that combine the available tools with self-developed procedures.

The effort required and the appropriate technology selection depends on the high availability goals you want to achieve.

Table 4 summarizes the techniques useful for specific scenarios.

Table 4 Summary of high availability scenarios

Scenario	Service level	Technique	Remarks
z/VM failure	Few minutes	Restart on zSeries server Automated start procedure If restart not successful site takeover cold standby	
Application takeover	Few minutes	Monitoring tool and heartbeat	Linux only solution
Data availability	Few minutes	Shared z/VM minidisks, stonith, heartbeat, monitoring tool, or distributed file system	Solution depends on used file system
Adapter failure	Continuous	z/VM VIPA or Linux router with OSPF	
z/VM failure	Continuous	Network Dispatcher, hot standby	
Application failure	Continuous	Load balancing cluster, LVS, hot standby	Application must support cluster
Data high availability	Continuous	Database on z/OS with Parallel Sysplex	

References

IBM Redbooks

Linux for IBM eServer zSeries and S/390: Distributions, SG24-6264

Linux on IBM eServer zSeries and S/390: ISP/ASP Solutions, SG24-6299

IBM eServer zSeries 900 Technical Guide, SG24-5975

IBM Total Storage Solutions for Disaster Recover, SG24-6547

TCP/IP Solutions for VM/ESA, SG24-5459

Implementing ESS Copy Services on S/390, SG24-5680

IBM eServer zSeries Connectivity Handbook, SG24-5444

OSA-Express Implementation Guide, SG24-5948

zSeries HiperSockets, SG24-6816

OS/390 MVS Parallel Sysplex Configuration Volume 1: Overview, SG24-2075

WebSphere Edge Server: Working with Web Traffic Express and Network Dispatcher, SG24-6172

z/VM Documentation

z/VM V4R2.0 Planning and Administration, SC24-5995

z/VM V4R1.0 System Operation, SC24-6000

z/VM V4R2.0 Virtual Machine Operation, SC24-6036

z/VM V4R2.0 CP Command and Utility Reference, SC24-6008

z/VM V4R1.0 CMS User's Guide, SC24-6009-00

z/VM TCP/IP Level 420 Planning and Customization, SC24-6019

z/VM TCP/IP Level 3A0 User's Guide, SC24-5982

z/VM V3R1.0 REXX/VM Users Guide, SC24-5962

z/VM V4R2.0 REXX/VM Reference, SC24-6035

VM RSCS General Information, GH24-5218

Other

Geographically Dispersed Parallel Sysplex: The Ultimate e-business Availability Solution, IBM Whitepaper GF22-5114-02

Network Dispatcher, V3.6: Scalability, Availability and Load-balancing for TCP/IP Applications, available from:

http://www-3.ibm.com/software/webservers/edgeserver/library.html

Referenced Web sites

Linux High Availability Project

`http://www.linux-ha.org`

Linux mon

`http://www.kernel.org/software`

Linux for zSeries Device Driver Manual

`http://www10.software.ibm.com/developerworks/opensource/linux390/documentation-2.4.7.shtml`

Zebra

`http://www.zebra.org`
`ftp://ftp.suse.com/pub/suse/i386/7.3/full-names/src/zebra-0.92a-50.src.rpm`

CPINT Tool

`http://linuxvm.org/penguinvm/programs/`

WebSphere

`http://www-3.ibm.com/software/webservers/appserv/doc/v35/ae/infocenter/was/030503.html`
`http://www-3.ibm.com/software/webservers/appserv/doc/v35/ae/infocenter/was/030503.html`

The team that wrote this paper

This paper was produced during a workshop and subsequent experiments on high availability scenarios. The workshop and experiments were performed at the Technical Marketing Competence Center in the IBM Lab in Boeblingen (Germany).

Erich Amrehn is a certified Senior IT Specialist at the EMEA Technical Marketing Competence Center (TMCC), Boeblingen, Germany. Before joining the TMCC, he worked as a project leader at the International Technical Support Organization, Poughkeepsie Center. During that time, he wrote redbooks and taught Linux topics worldwide. Before joining the ITSO in 1998, he worked as a technical consultant to the IBM System/390 division for e-commerce on S/390 in Europe, the Middle East, and Africa. He also has 13 years of VM experience in various technical positions in Germany and other areas in Europe and worldwide.

Ronald Annuss is an IT Specialist working for IBM in Germany. He holds a diploma in Geodesy from the Technical University Berlin. He has worked with Linux for S/390 since it became available in early 2000, and has worked on several Linux for S/390 projects with customers in Germany. He is an Red Hat Certified Engineer (RHCE).

Bernhard Dierberger is an IT specialist at the IBM Technical Marketing Competence Center (TMCC), in Boeblingen (Germany). He holds a Doctoral degree in physics from the University of Tuebingen(Germany). Before joining the IBM lab in Boeblingen, Bernhard was a technical consultant for IBM Global Services Germany. Since 1999 his work at the IBM development lab has focused on J2EE and WebSphere technologies. Currently he is involved in several customer Linux for zSeries Proof of Concept projects running at the TMCC.

Richard Plasun: is a in IBM Austria. He holds a Doctoral degree in Electrical Engineering from Technical University Vienna, Austria. Since joining IBM, he has specialized on the zSeries platform, especially on Linux for zSeries. Currently, he is involved in several Linux for zSeries Projects in Austria.

U. Sager: is a Consulting IT Specialist working on the IBM S/390 Technical Support Team in Germany for IBM EMEA Central Region. He has worked in the OS/390 area for 15 years, mainly as a systems engineer and technical support specialist for large systems. More recently he has specialized in OS/390 e-business. His areas of expertise include z/OS UNIX System Services, Linux for zSeries, e-business infrastructure, Java and e-business connectors. He holds a Master of Science Degree from the University of Manchester Institute of Science and Technology, England. He has co-authored several IBM Redbooks including "Connecting DB2 for OS/390 to the Internet" and "WebSphere Customization and Usage." He is currently involved in several Linux for zSeries projects in Germany.

Thanks to following people to their invaluable contributions to this paper:

F. Kirschner, U. Pimiskern, E. Puritscher

Appendix

USER DIRECTORY of our z/VM System

```
.........
USER LNX31  PASSWORD 128M 1024M
ACCOUNT  ENLLINUX
MACHINE  ESA
IUCV ALLOW
IUCV ANY
IPL CMS
CPU 00
CPU 01
CONSOLE  009 3215
SPOOL 000C 2540 READER *
SPOOL 000D 2540 PUNCH A
SPOOL 000E 1403 A
LINK MAINT 0190 0190 RR
LINK MAINT 019D 019D RR
LINK MAINT 019E 019E RR
LINK LINUX390 191 191 RR
* dedicated disks
  DEDICATE 500 520A
  DEDICATE 501 520B
* ctc connctions
  SPECIAL 600 CTCA
  SPECIAL 601 CTCA
  SPECIAL 602 CTCA
  SPECIAL 603 CTCA
  SPECIAL 604 CTCA
  SPECIAL 605 CTCA
  dedicate 610 6740
dedicate 611 6741
* osa devices
  DEDICATE A02 A02
  DEDICATE A03 A03
  dedicate 3002 3002
  dedicate 3003 3003
MDISK 0150 3390 0001 3338 FR5208 MR ONE4ME TWO4ME THR4ME
MDISK 0151 3390 0001 3338 FR5209 MR ONE4ME TWO4ME THR4ME
MDISK 0250 3390 0001 3338 FR5218 MW ONE4ME TWO4ME THR4ME
MDISK 0251 3390 0001 3338 FR5219 MW ONE4ME TWO4ME THR4ME
*
USER LNX32  PASSWORD 128M 1024M
ACCOUNT  ENLLINUX
MACHINE  ESA
IUCV ALLOW
IUCV ANY
IPL CMS
CPU 00
CPU 01
CONSOLE  009 3215
SPOOL 000C 2540 READER *
SPOOL 000D 2540 PUNCH A
SPOOL 000E 1403 A
LINK MAINT 0190 0190 RR
LINK MAINT 019D 019D RR
LINK MAINT 019E 019E RR
LINK LINUX390 191 191 RR
* dedicated disks
```

```
  DEDICATE 500 520C
  DEDICATE 501 520D
* ctc connctions
  SPECIAL 600 CTCA
  SPECIAL 601 CTCA
  SPECIAL 602 CTCA
  SPECIAL 603 CTCA
  SPECIAL 604 CTCA
  SPECIAL 605 CTCA
  dedicate 610 6742
  dedicate 611 6743
* dedicate 3002 3002
* dedicate 3003 3003
MDISK 0150 3390 0001 3338 FR5208 MR ONE4ME TWO4ME THR4ME
MDISK 0151 3390 0001 3338 FR5209 MR ONE4ME TWO4ME THR4ME
MDISK 0250 3390 0001 3338 FR5218 MW ONE4ME TWO4ME THR4ME
MDISK 0251 3390 0001 3338 FR5219 MW ONE4ME TWO4ME THR4ME
```

mon init script

```
#!/bin/sh
#
# start/stop the mon server
#
# You probably want to set the path to include
# nothing but local filesystems.
#
# chkconfig: 2345 99 10
# description: mon system monitoring daemon
# processname: mon
# config: /etc/mon/mon.cf
# pidfile: /var/run/mon.pid
#
PATH=/bin:/usr/bin:/sbin:/usr/sbin
export PATH

# See how we were called.
case "$1" in
    start)
        echo -n "Starting mon daemon: "
        /usr/lib/mon/mon -f -c /etc/mon/mon.cf -b /usr/lib/mon
        echo
        touch /var/lock/subsys/mon
        ;;
    stop)
        echo -n "Stopping mon daemon: "
        killproc mon
        echo
        rm -f /var/lock/subsys/mon
        ;;
    status)
        status mon
        ;;
    restart)
        killall -HUP mon
        ;;
    *)
        echo "Usage: mon {start|stop|status|restart}"
        exit 1
esac
```

```
exit 0
```

lnx33/ospfd.conf

```
!
! ospfd configuration for lnx33
!
hostname lnx33
password zebra
enable password zebra
log file /var/log/ospfd.log
log record-priority
service advanced-vty
!
!
interface lo
!
interface ctc0
!
interface ctc1
!
router ospf
 ospf router-id 10.33.0.1
 passive-interface lo
 network 10.31.0.1/32 area 1
 network 10.32.0.1/32 area 1
!
! to limit the access of the configuration-console to localhost:
access-list term permit 127.0.0.1/32
access-list term deny any
!
line vty
 access-class term
!
```

lnx33/zebra.conf

```
!
! Zebra configuration for lnx33
!
hostname lnx33
password zebra
enable password zebra
log file /var/log/zebra.log
log record-priority
service advanced-vty
!
!
interface lo
!
interface ctc0
!
interface ctc1
!
! to limit the access of the configuration-console to localhost:
access-list term permit 127.0.0.1/32
access-list term deny any
!
```

```
line vty
 access-class term
!
```

Notices

This information was developed for products and services offered in the U.S.A.

IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to:
IBM Director of Licensing, IBM Corporation, North Castle Drive Armonk, NY 10504-1785 U.S.A.

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law: INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

COPYRIGHT LICENSE:
This information contains sample application programs in source language, which illustrates programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs. You may copy, modify, and distribute these sample programs in any form without payment to IBM for the purposes of developing, using, marketing, or distributing application programs conforming to IBM's application programming interfaces.

This document created or updated on June 21, 2002.

Send us your comments in one of the following ways:

- Use the online **Contact us** review redbook form found at:
  ibm.com/redbooks
- Send your comments in an Internet note to:
  redbook@us.ibm.com
- Mail your comments to:
  IBM Corporation, International Technical Support Organization
  Dept. HYJ Mail Station P099
  2455 South Road
  Poughkeepsie, NY 12601-5400 U.S.A.

Trademarks

The following terms are trademarks of the International Business Machines Corporation in the United States, other countries, or both:

Redbooks(logo)™
AFS®
AIX®
CICS®
DB2®
DB2 Connect™
ECKD™
Enterprise Storage Server™
ESCON®
FICON™
FlashCopy®
GDPS™
Geographically Dispersed Parallel Sysplex™
IBM®
OS/390®
Parallel Sysplex®
Redbooks™
S/390®
SP™
VM/ESA®
Wave®
WebSphere®
z/OS™
z/VM™
zSeries™

The following terms are trademarks of International Business Machines Corporation and Lotus Development Corporation in the United States, other countries, or both:

Lotus®
Word Pro®

The following terms are trademarks of other companies:

ActionMedia, LANDesk, MMX, Pentium and ProShare are trademarks of Intel Corporation in the United States, other countries, or both.

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

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

C-bus is a trademark of Corollary, Inc. in the United States, other countries, or both.

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

SET, SET Secure Electronic Transaction, and the SET Logo are trademarks owned by SET Secure Electronic Transaction LLC.

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