IBM DB2 UDB and SteelEye LifeKeeper for Linux- A High Availability Database Solution

Andy Beaton
IBM Toronto Lab

1) Introduction

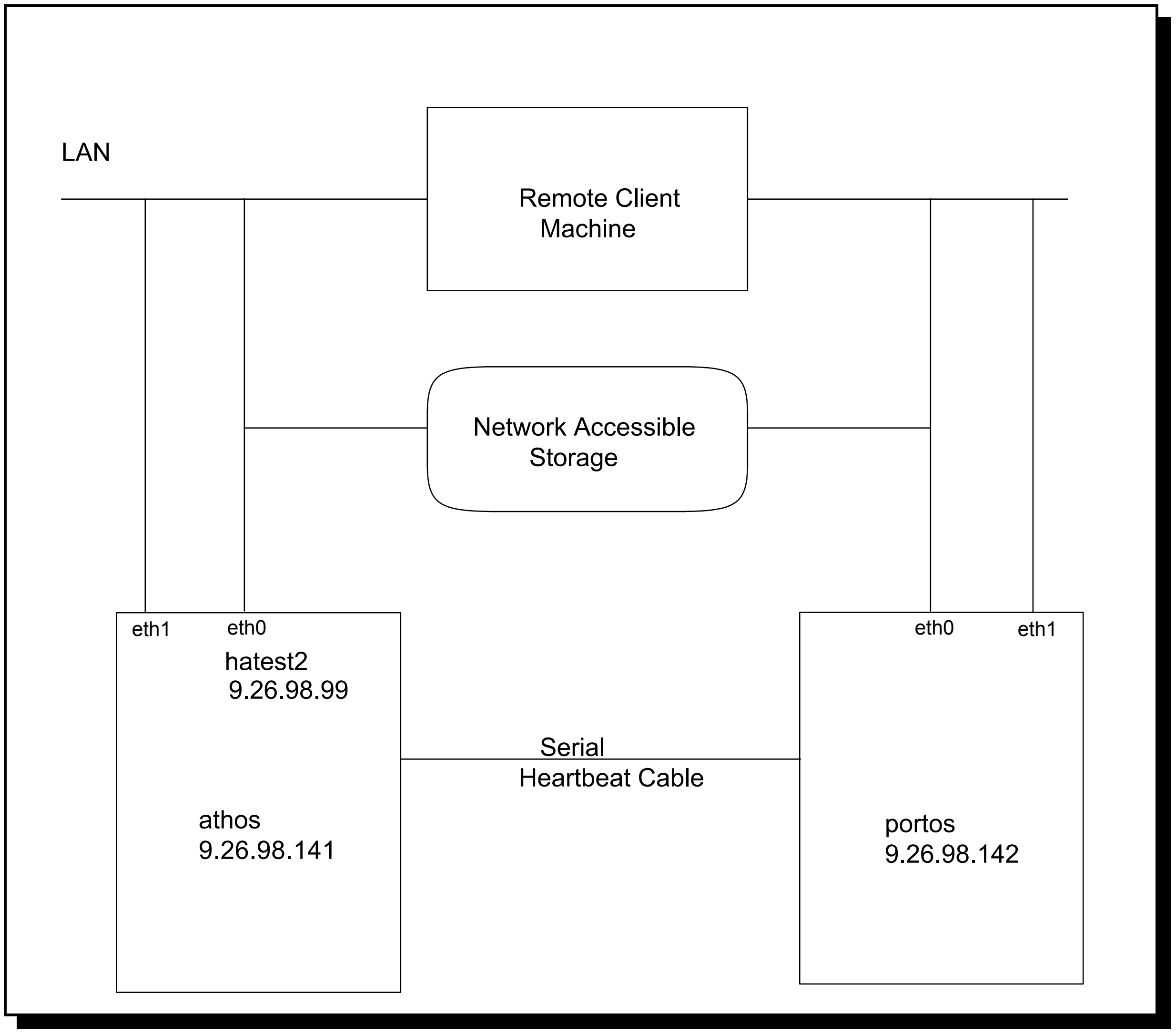
This paper provides an introduction and sample configuration instructions for running IBM® DB2® Universal Database™ (DB2 UDB) on a cluster of Linux servers, with high availability capabilities provided by SteelEye LifeKeeper for Linux. The purpose of high availability software is to provide the user of the data on the server with near-continuous service, where any failure in the server or the software is automatically corrected for. We have used, when testing the procedures outlined in this paper, two IBM xSeries® 335 Linux servers running RedHat Linux 8.0 with the 2.4.18 kernel, DB2 UDB V8.1 for Linux as our database solution, and SteelEye LifeKeeper 4.2 as our high availability clustering solution.
A properly designed high availability cluster will be able to continue to provide service after an error in any single point of failure, with no more than a minimum period of unavailability as the software automates the takeover of resources from the failed machine onto a backup machine.

In its simplest form, a high availability cluster consists of two servers with shared disk space between them and an extra connection between the two servers to provide a heartbeat between the two machines. On one of the servers (Server A in this example) is where we will run our database. In the event of a problem with Server A, the second server, Server B, will take over running the database, take over the IP address that the users are using to connect to server A, and resume database operations with the minimum of disruption to the users. After a brief failover, the length of which will depend on the nature of the failure and the size and state of the database at the time of failure, the user will once again be able to access the data from the same database at the same IP address. Server B will continuously monitor the state of Server A (through the LAN, and for redundancy, through a separate serial cable) in order to know when and if to take over the resources

The servers that were used in setting up and testing this product were a pair of IBM xSeries 335 servers running RedHat Enterprise Linux Server 3. Wherever possible, redundant components were used to provide protection from component failures. Each machine contains two Network Interface Cards (NIC). The software will also work with a single NIC, but this is a less than optimal solution in that a NIC failure will result in a failover, rather than an instantaneous network card takeover. Both servers are also connected by a null modem cable between the RS-232 serial ports. This serial cable will provide a path for a heartbeat signal between the two servers. Without a redundant signal path between the two servers, the backup server will be unable to distinguish between a communications failure and a server failure, and may try to take over a still active database. Remote clients were tested from an IBM RS6000®/P660 server running AIX® 5.1.0.

This is the special user ID that can be used to develop programs in a special fenced area of memory. When the user ID's have been created, then the ownership can be set for the directories using the following commands;

```
chown svtdbm /home/svtdbm
chown db2fenc1 /hahome/db2fenc1
chgrp db2grp /home/svtdbm /home/db2fenc1
passwd svtdbm (you will be prompted to enter a password).
```



It is important that the UID and GID numbers for the instances match on both of the servers; otherwise errors could occur when the high availability software is attempting to take over resources after a failover.

3) DB2 UDB Setup

DB2 UDB will need to be installed on each of the nodes of the cluster. This can be done through either the db2setup utility or the db2_install utility.

If you are using the db2_install utility, the installer will ask which product is to be installed; specify DB2 ESE and press <enter>. All necessary files will be installed.

The db2setup utility is started on the console command line. It will ask the person doing the installation if an administrator and if a user instance should be automatically created. It is easiest to answer "no" here, and do instance creation afterwards using the command line.

After DB2 UDB has been installed, the user instance can be created by root with the following command:

```
#cd /opt/IBM/db2/V8.1/instance
# bash db2icrt -u db2fenc1 svtdbm
```

where the -u flag sets the ID to be used when developing in a fenced section of memory.
After the instance has been created, it will be necessary to unmount the home filesystem, mount it on the second server, remove the ~/sqllib directory and create the instance on the second server using the same commands.

4) LifeKeeper Setup

The instructions presented here are a brief overview of the SteelEye LifeKeeper installation instructions; for full details, refer to *LifeKeeper for Linux Planning and Installation Guide*. Software can be purchased from SteelEye, or downloaded from http://www.steeleye.com
The installation will need to be done separately on each server member of the cluster.

The first CD to be mounted (orthe first .tar file to download) contains the installation files for the operating system; in this case, we are using the files in the Installation Support file set de_rh30.tar.gz .Within this file set is a setup script, which when invoked at the command line, will install the necessary files for LifeKeeper to work with any given version of Linux.
You will be prompted for the addition of a number of components. Some will already be installed.

a) The LifeKeeper Distribution Enabling Package is required. It has an undocumented dependency; ncurses4 >= 5.0-2 is required, which can be checked with the command **rpm -qa | grep -i ncurses4**. If it is missing, or not at a high enough level, the RPM can be downloaded from www.redhat.com.

b) The Java Runtime Environment is required to run the GUI administration tool locally.

c) As we are using Network Storage for our shared storage needs, we have answered ‘no’ to creating additional SCSI disk nodes, and to probing LUNs on SCSI devices.
Our system had kernel version 2.4.21-9 installed, so we were not prompted to upgrade the kernel.

d) We did not have Data Replication or NFS RPMs available, nor did we have any need for them, so we skipped the next two installation steps. If either of these is required, follow one of these solutions:

1) Apply the LifeKeeper kernel modules and allow the LifeKeeper setup routine to create a new initial ramdisk.. The setup routine will also alter the grub boot configuration to use this new ramdisk.
.
2) Reboot the system with a different kernel and rerun setup to apply the necessary RPMs for that kernel.

e) Download and install the LifeKeeper DB2 Application Recovery Kit from www.steeleye.com; SteelEye-lkDB2-4.2.0-3.i386.rpm

f) Finally, the licensing package, which is required, is installed.
At the end of this process, you will be provided with a unique Host ID for the server being configured. This Host ID needs to be sent to SteelEye (http://licensing.SteelEye.com/permanent/), who will then provide a permanent license key. Note that these Host IDs and license keys are keyed to individual machines. If you want to install LifeKeeper on a different machine, a new Host ID and license key will need to be obtained. Temporary license keys can also be acquired for evaluation purposes.

When SteelEye has provided the license key, it needs to be entered into the system using the command **/opt/LifeKeeper/bin/lkkeyins**.
You will be prompted for a file containing the license key, or asked to insert the license key at the command prompt.

The second CD, or the second set of files to be downloaded contains the LifeKeeper for Linux installation files. The file set we have downloaded is lk.tar.gz. When uncompressed on the server, it contains
SteelEye-lk-4.2.0-6.i386.rpm
SteelEye-lkIP-4.2.0-6.i386.rpm
SteelEye-lkCCISS-4.2.0-6.i386.rpm
SteelEye-lkMAN-4.2.0-6.i386.rpm
SteelEye-lkGUI-4.2.0-6.i386.rpm
SteelEye-lkRAW-4.2.0-6.i386.rpm
SteelEye-lkHLP-4.2.0-6.i386.rpm

The RPMs that are relevant to us are SteelEye-lk-4.2.0-6 (the core LifeKeeper software), SteelEye-lkIP-4.2.0-6 (software to control IP address takeover), SteelEye-lkMAN-4.2.0-6 (data for the online Linux man help utility), SteelEye-lkGUI-4.2.0-6 (the GUI administration tool) and SteelEye-lkHLP-4.2.0-6 (the online browser help service). The command **rpm -i <filename>** will install the files. We can confirm that the packages have been correctly installed with the command **rpm -qa | grep -i <package name>.**

In order to use the man utility to provide information about LifeKeeper, the system parameter MANPATH needs to modified as follows:
MANPATH=/opt/LifeKeeper/man:$MANPATH
export MANPATH

The LifeKeeper documentation asks that LKROOT be defined as /opt/LifeKeeper .
We have added this to our .profile in the root home directory. Doing so will simplify configuration.

When both server nodes have had all of the necessary software installed and configured, we will connect the two servers with an RS-232 9-pin null modem cable between the two serial ports. This cable will provide a redundant heartbeat signal between the two server nodes. If there is a loss of communication over the LAN, the serial cable heartbeat signal will be able to confirm the failure of a server, or instead signify the failure of the LAN.

Testing the functionality of the serial heartbeat cable is done as follows:
From the root account of server 1, enter the following command:
$LKROOT/bin/portio -r -p port -b baud
On our instance, the port was /dev/ttyS0, and the baud rate was 9600 (which are typical default values), so we entered
$LKROOT/bin/portio -r -p /dev/ttyS0 -b 9600

From the second server, enter the command **echo Helloworld | $LKROOT/bin/portio -p port -b baud**, where once again port is the local communications port and b is the baud rate.
If the test is successful, the message “Helloworld” should be written to the console of the first server. Further details can be found in the *LifeKeeper for Linux Planning and Installation Guide.*

To start LifeKeeper on both servers, log on as root and run the command
$LKROOT/bin/lkstart
This is necessary to protect the resources on the servers.
To start the GUI server backend processes, log on as root and run the command
$LKROOT/bin/lkGUIserver start
This is used to administer the resource protection. Once the administration tasks have been completed, it is no longer needed, but should be left running in case further remote adminstration becomes necessary.

In order to allow a client full access to the database resource regardless of the server state, we will assign an IP address to the cluster that will be used at all times for client access. We have set up an address 9.26.98.99, to which we have assigned the name hatest2 (hatest2.torolab.ibm.com is our fully qualified name).

We have set up the following lines in /etc/hosts on both servers:

```
9.26.98.141          athos.torolab.ibm.com  athos
9.26.98.142          portos.torolab.ibm.com  portos
9.26.98.99          hatest2.torolab.ibm.com hatest2
```

Creating a Database:

Connect to the primary server using the instance ID you have selected, which in our case, is db2inst1.
Once connected, issue the command db2start. Once the server has been started, it needs to be configured for client server access. Ports need to be specified in /etc/services on both machines to allow remote client machines to connect to the server: here we have configured a connection port and an interrupt port.

```
xdb2inst1          18806/tcp
xdb2inst1_int        18807/tcp
```

The database server is then configured to use this port with the commands
db2 update dbm cfg using svcename xdb2inst1
db2set db2comm=tcpip

The changes are then committed by issuing **db2stop** and **db2start**.
We then create our test database with the command **db2 create db test1**.

From the client machine, we catalog our test database with the following commands:
db2 catalog tcpip node hatest2 remote hatest2 server 18806
db2 catalog db test1 at node hatest2 authentication server

The database connection can be tested with the command
db2 connect to test1 user db2inst1 using <password>.

The IP address hatest2 is attached to the server <athos>. The servers are communicating over the LAN and over the Serial Heartbeat Cable. From the remote client machine, we can use the GUI tool to administer the servers and the resources assigned to the servers; the resources we have to assign are the IP address, the database server and the shared storage.

Before the GUI Administration Tool can be used, the Java policy file needs to be copied to the client machine. The file <http://athos:81/java.policy> can be saved to the users' home directory on the client machine as .java.policy (note the periods).

To start the GUI administration tool, connect to the HTML server at port 81 on the server: <http://athos:81> from a Java-enabled Web browser. If Java has not been installed on the client machine, it needs to be. The Java Runtime Engine 1.3.1 is the minimum level needed for the GUI Administration client to work.

At this point, we can log onto the GUI server with the “Connect” command, under the File menu in the GUI Adminstration Client. When requested, enter the root ID and password in the spaces provided.

Once logged in, we can proceed to configure the resources necessary to protect the cluster.

Start by configuring the communication paths to the secondary server. The configuration is invoked under the Edit menu-->Server-->Create Comm Path. Two communication paths need to be created, one path over TCP/IP, and one path over a serial cable. In order to create the TCP/IP path, the following information is needed, to be filled in when prompted: both server names, IP addresses for both servers (not the shared address), and the priority to be assigned to the initial configuration. In order to create the serial communication phath, the following information must be entered at the prompts: the server names, the serial devices on both servers used for the connection, and the baud rate.

Once the communication paths have been established, resource hierarchies are created. In the simple intance we are setting up here, 3 resource hierarchies are required: a database instance, a shared IP address, and a shared home filesystem.

To create the shared DB resource, run Edit--> Resource--> Create Resource Hierarchy, and specify the following values:

Select Recovery Kit	-->	DB2 Database
Switchback Type	-->	Intelligent
Server	-->	<Primary server name>
DB2 Instance	-->	db2inst1 (or other preset instance)
Database Tag	-->	DB2 db2inst1

Check the information reported back, and click Continue.

Target Server	-->	<Backup server name>
Switchback Type	-->	Intelligent
Template Priority	-->	1
Target Priority	-->	10
DB2 Instance (Information Only)	-->	db2inst1
Database Tag	-->	DB2 db2inst1

The creation of the DB2 resource will automatically create a child resource for the shared home filesystem.

The next resource to be created is the shared IP resource. Once again, we run Edit-->Resource-->Create Resource Hierarchy, and specify the following values:

Select Recovery Kit	-->	IP
Switchback Type	-->	Intelligent
Server	-->	<Primary Server Name>
IP Resource	-->	<Shared IP address in dotted quad form>
Netmask	-->	<As specified by your Network Admin>
Network Interface	-->	eth0
Backup Interface	-->	eth1
IP Resource Tag	-->	ip-9.26.98.99

Check the information reported back, and clixkContinue.

Target Server	-->	<Backup Server Name>

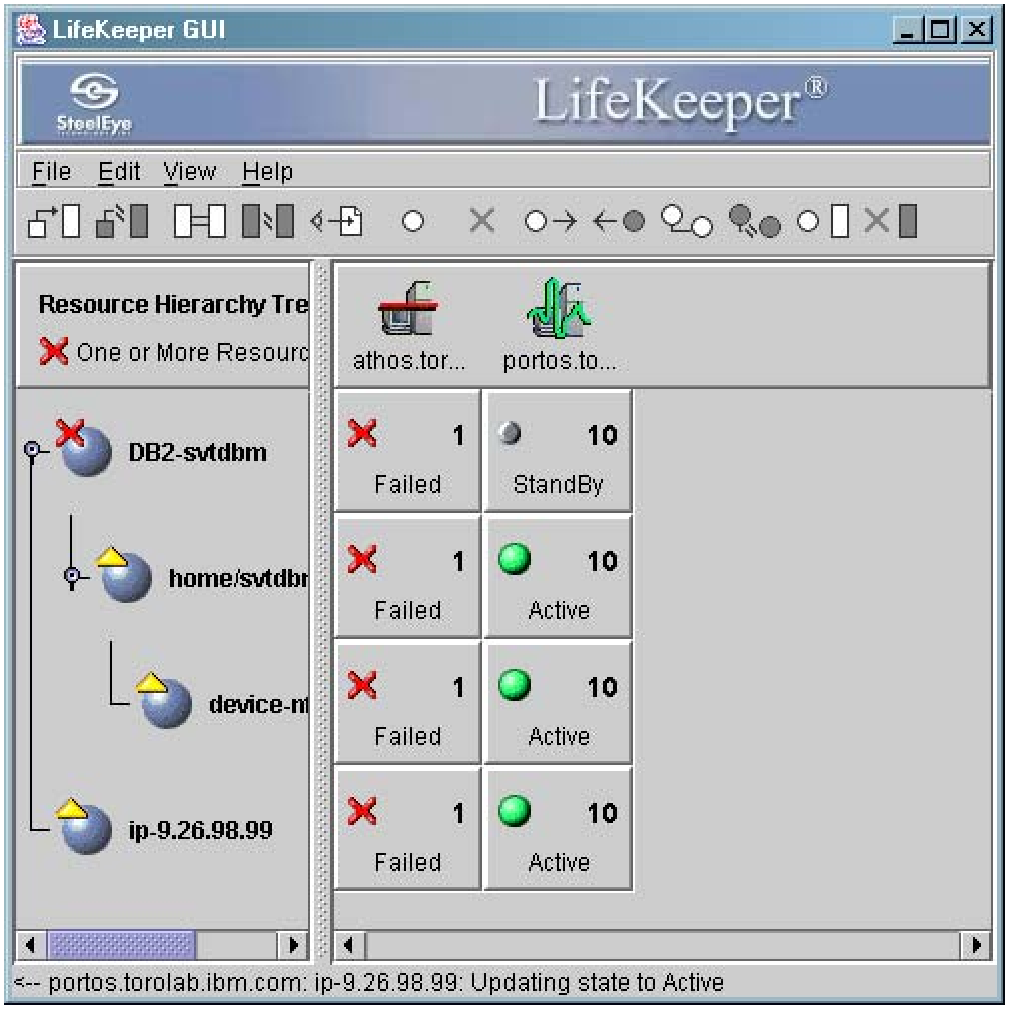
Switchback Type	-->	Intelligent
Template Priority	-->	1
Target Priority	-->	10

Accept the defaults for IP Resource and Netmask. On the Target Server,

Network Interface	-->	eth0
Backup Interface	-->	eth1
Resource Tag	-->	ip-9.26.98.99

and press <Extend> to protect the resource over both servers.

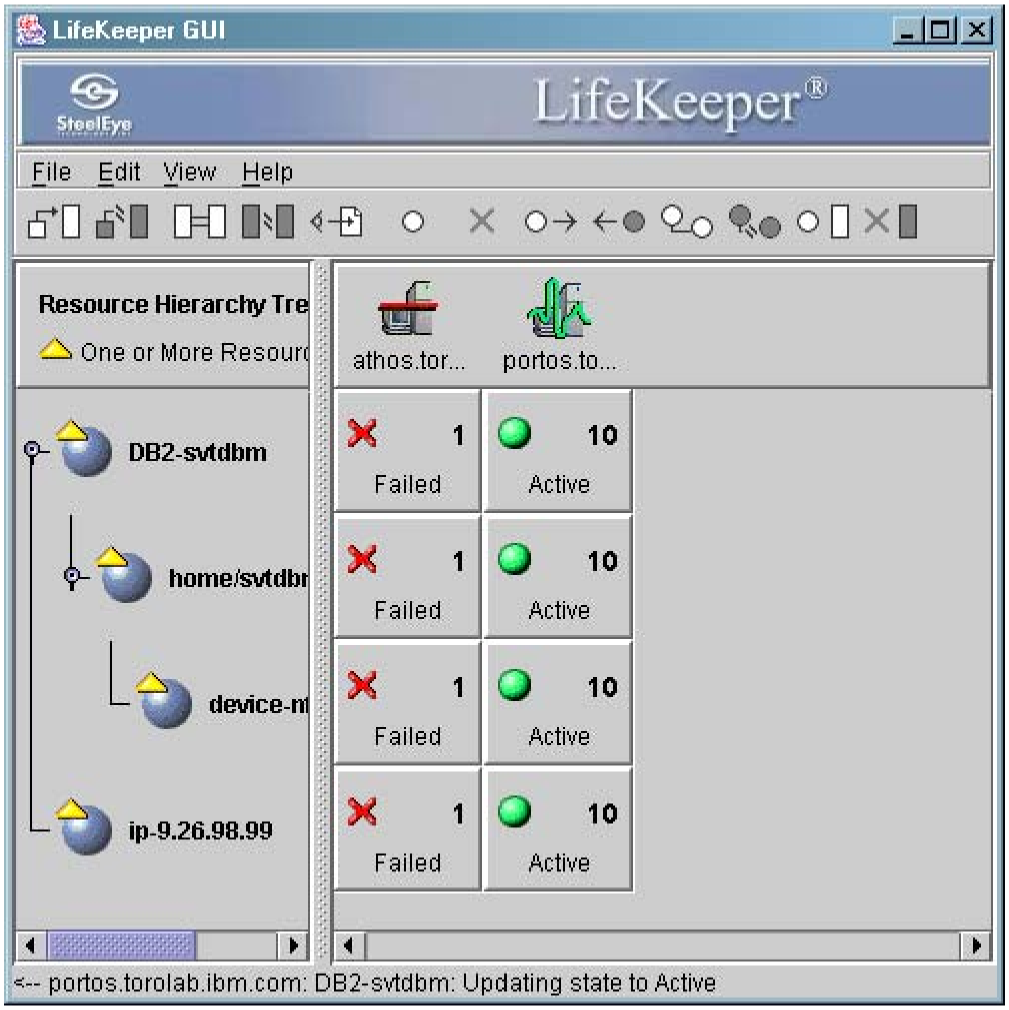
At this point, we can force a failure on the server node athos: if we issue a shutdown command on athos, it will simulate a fatal hardware or software failure. The failover can be monitored from the client machine using the GUI Administration Tool. In the illustration below, a shutdown -r now command has been issued on the server athos. The server portos is in the process of taking over the resources, and has mounted the shared filesystem and assumed the shared IP address:



Resource	athos.tor...	portos.to...
DB2-svtdbm	1 Failed	10 StandBy
home/svtdbr	1 Failed	10 Active
device-n	1 Failed	10 Active
ip-9.26.98.99	1 Failed	10 Active

```
<-- portos.torolab.ibm.com: ip-9.26.98.99: Updating state to Active
```

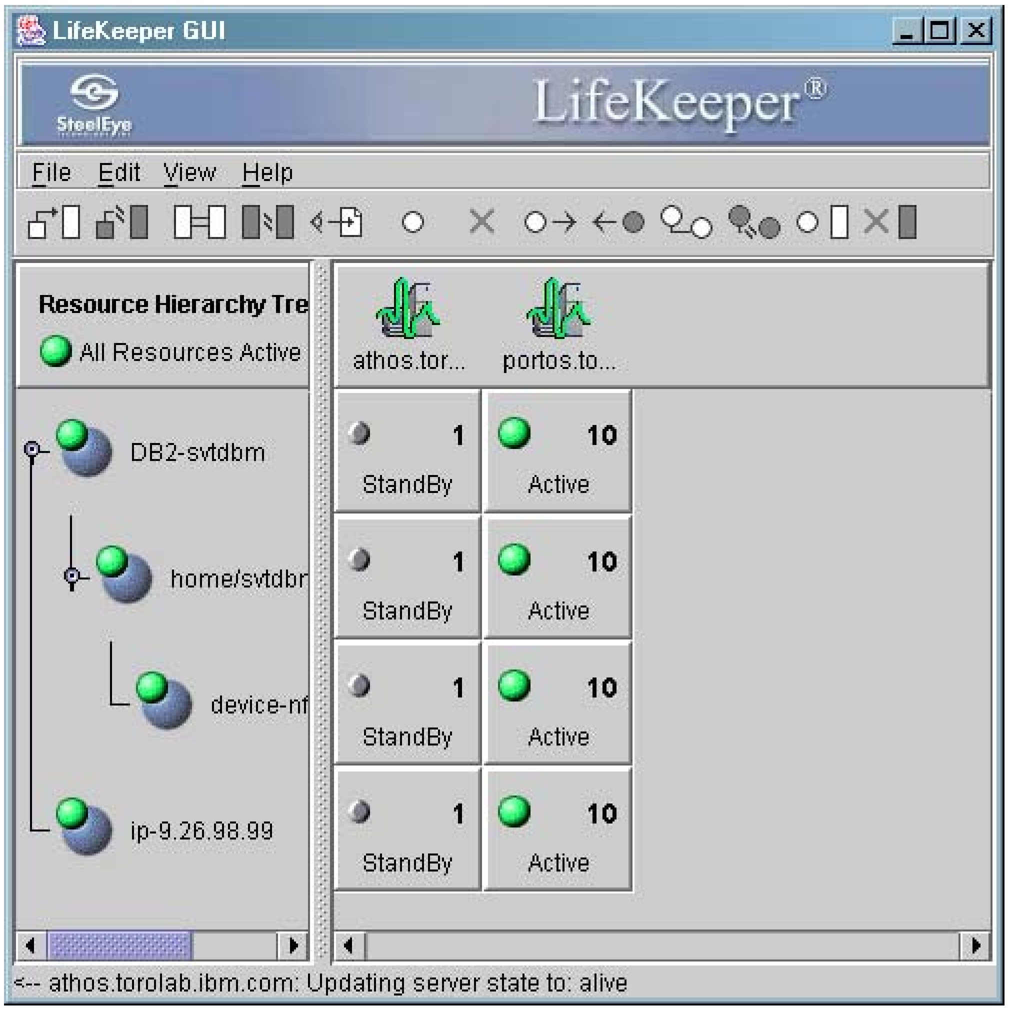
At this point, shortly after the failure of the node athos, the shared storage and the IP address have failed over to the node portos. The DB2 UDB database resource is in the process of starting up and undergoing internal consistency checks. Within a few minutes (depending on the size of the database and its state at the time of failure), the database will also become active on the node portos.



<-- portos.torolab.ibm.com: DB2-svtdbm: Updating state to Active

At this point, the failover is complete. Remote clients can now connect and intereact with the database with no evidence that any error took place. When the failed server is returned to service, the resources will either restart on the original server, or continue to run on the backup server depending on whether the cluster resources have been set with automatic switchback on or with intelligent switchback on. With automatic switchback, the resources return to the original server. With intelligent switchback, manual intervention is necessary to return the cluster to its original state.

We accepted the intelligent switchback settings when we were configuring the cluster. Therefore, after the failed node has been returned to service, the GUI Administration Tool looks like this:



	athos.tor...	portos.to...
DB2-svtdbm	1 StandBy	10 Active
home/svtdbr	1 StandBy	10 Active
device-nf	1 StandBy	10 Active
ip-9.26.98.99	1 StandBy	10 Active

<-- athos.torolab.ibm.com: Updating server state to: alive

5) Testing DB2 UDB and LifeKeeper

Implementing a Highly Available cluster is a three-stage process. The stages are as follows:
1. Carefully plan the cluster.
2. Implement the cluster.
3. Test the cluster. If the test results are not satisfactory, return to step 1.

Planning the proper setup of a Highly Available cluster is not within the scope of this document. We are able to provide some testing guidelines, though.

Testing the cluster is as important as setting it up, to make sure that it behaves as expected. Ideally, every single point of failure that has the potential to bring down a cluster should be tested. Granted, it may not be practical to shut down a building’s power supply in order to test backup power supplies, but a single source of electricity is a single point of failure, and until the system has been physically tested, you will never be entirely certain that the cluster will behave in practice as it does in theory.
The best testing environment is on the cluster itself. Enough scheduled downtime should be set aside to thoroughly test the system before the system is put into production. A short, planned outage is preferable to a long, unplanned outage that reveals that an untested point of failure has left an application unavailable.
The testing procedure itself is simple. Connect to the cluster from a client machine, cause one of the points of failure to fail, and watch to ensure that the failover takes place properly, and that the application is available and properly configured after failover. If the cluster is built using automatic switchback configuration, check again after service has been restored to the original node. If the cluster is built using intelligent switchback configuration, bring up the original node again, then cause the second node to fail, which should restore the system to its original node.
When testing the availability of the application, be sure that accounts and passwords work as expected, hostnames and IP addresses work as expected, the data is complete and up-to-date, and the changeover is essentially transparent to the user.
Configure a remote machine to be able to connect to the highly available DB2 UDB database. A script can be easily written that will connect to our database, select some data from a table, record the results, and disconnect from the database. If these steps are set inside a loop that will run until interrupted by the operator, the procedure can be used to monitor the state of the cluster. Keep in mind that the script should continue even if the database cannot be contacted. This way, when the database restarts, it will provide a benchmark for the length of time that failover is expected to take.

```
while :
do
db2 connect to database
```

```
db2 "select count(*) from syscat.tables"
db2 connect reset
sleep 60
done
```

Points of failure that should be considered and tested are:
1. Software installation errors
2. Hardware configuration errors
3. Power failures
4. Network failures, both in hardware and software
5. Hardware failures in the CPU, the disk drives, and any of the physical infrastructure
6. Careless operator behavior
7. Software failures in the operating system, LifeKeeper, or the applications

This is not an exhaustive list. Cluster administrators are in the best position to know where points of failure are in their own clusters.

1. Software installation errors.
The first test for correctly installed software is starting the software. After installation, check the error log to ensure that no major errors occurred.

2. Hardware configuration errors.
There are a number of tests to execute to ensure that the hardware is correctly configured:
- Use the date command on all nodes to ensure that they are synchronized.
- Use ifconfig <device> to check the network adapters.
- Use netstat -i to check the network configuration.

3. Power failures.
Testing failover behavior in various types of power failure situations is worthwhile to ensure that the cluster is physically set up to behave in the manner desired. Testing for power failures in individual components can be accomplished by simply pulling plugs out of sockets, or hitting power buttons. Larger power failures can be tested by throwing switches in the building's electrical panel (Make sure the person doing this knows what he or she is doing!).

4. Network failures.
Test the network starts by ensuring that the network behaves as it is expected to upon startup. Be sure that rlogin will connect to all service addresses from all cluster nodes.
Network failures tend to be of two kinds: hardware and software. Hardware failures can be tested by physically unplugging network cables. One at a time, unplug every network cable entering the cluster machines. Plug each back in again before removing the next plug. Remember to check the RS-232 serial cable, or, if the cluster is also using SSA or SCSI devices as its serial connection , unplug them one at a time. This will simulate the

physical failure of individual network controllers, or problems with the cables themselves. A properly designed and implemented system will be able to survive the loss of any one of these components.
A network software failure can be simulated by killing network processes running on the primary server. The **ps -ef | grep <id>** command will provide a list of running processes, including their group and PID. Selecting a likely looking process from the TCP/IP group and issuing a **kill -9 <PID>** command will simulate the failure of the network software. Killing processes such as /usr/sbin/inetd or /usr/sbin/portmap will be good tests.
Networking software can also be halted using Linux's built-in administration tools.

5. Hardware failures.
Hardware failures can be simulated by a variety of brute force methods. CPU failures can be mimicked by pushing power or reset switches, by killing processes, or corrupting the memory. Issuing the following command from the root user account is a good way to bring down a machine:
```
# echo “hello world” > /dev/kmem
```
A good way to test for hardware failures in the DASD is to pull power cables, or physically pull disks out of their cabinets.

6. Careless operator behavior.
Careless operator behavior can result in any of the situations outlined and tested in steps 4, 5, and 7.

7. Software failures.
Software failures can be simulated by killing critical processes, which also works to simulate many operator errors. Doing the
same with the db2sysc process will ensure that the scripts are properly configured to catch a failure in the critical application and will restart the application in the appropriate manner.

Not enough emphasis can be placed on the importance of carefully planning and testing the implimentation of a high availability cluster. With careful consideration of possible error paths and rigorous testing to ensure that all of the safeguards are behaving as expected, potential problems can be resolved long before they are seen by users. A properly set up cluster can be a great asset in allowing users to rely upon critical resources being available.

Trademarks

==