October 2003

DB2 Information Management Software

Highly Available DB2 Universal Database using Tivoli System Automation for Linux

Setup & Policies

Authors:

Enrico Joedecke
Steve Raspudic
Andy Beaton

Contents

1. Overview

1.1 Objectives

In this paper, we describe the implementation and design of highly available IBM DB2 Universal Database™ (DB2) environments in conjunction with the IBM Tivoli System Automation for Linux platform (TSA). Also included is a detailed description of the DB2 agent for TSA. We provide guidance and recommendations for highly available strategies using both DB2 Universal Database Enterprise Edition v7.1 and DB2 Universal Database v8.1. Practical considerations regarding design, implementation, testing, and maintenance work with the system are also discussed.

1.2 Overview

Chapter 2 lists the DB2 requirements for High Availability (HA) when used in conjunction with TSA. The requirements of TSA to automate DB2 are discussed in Chapter 3. The setup of a two node cluster running one DB2 instance automated by TSA is described in detail in Chapter 4. There is also a discussion of testing and maintenance issues.

1.3 Overview: DB2

IBM DB2 Universal Database (DB2) is the industry's first multimedia, Web-ready relational database management system, powerful enough to meet the demands of large corporations and flexible enough to serve medium-sized and small e-businesses. DB2 Universal Database combines integrated power for business intelligence, content management, and e-business with industry-leading performance and reliability. This combination, coupled with IBM Tivoli System Automation for Linux (TSA), strengthens the solution by providing a highly available computing environment. For more information about the IBM Data Management portfolio, please visit *www.software.ibm.com/data*

1.4 Overview: IBM Tivoli System Automation for Linux (TSA)

IBM Tivoli System Automation for Linux is a product that provides high availability by automating the control of resources such as processes, file systems, IP addresses and other arbitrary resources in Linux-based clusters. It facilitates the automatic switching of users, applications, and data from one system to another in the cluster after a hardware or software failure. A complete High Availability (HA) setup includes many parts, one of which is the HA software. As well as tangible items such as hardware and software, a good HA solution includes planning, design, customizing, and change control. An HA solution reduces the amount of time that an application is unavailable by removing single points of failure. For more information please visit *www.software.ibm.com/tivoli/products/sys-auto-linux*.

2. DB2 in an HA environment

In a typical DB2 setup, a single DB2 instance is running on a server. This DB2 instance has local access to data (its own executable image as well as databases owned by the instance). If this DB2 instance is made accessible to remote clients, an (unused) IP address must be assigned to this DB2 instance.

The DB2 instance, the local data, and the IP address are considered resources, whose control can be automated by TSA. Since these resources are closely related (for example, they collectively run on the same node at the same time), they are referred to as a *resource group*.

The entire resource group is collocated on one node in the cluster. In the case of a failover the entire resource group is started on another node. There are dependencies between the resources in the group:

- The DB2 instance must be started after, and stopped before, the disk resource.
- If remote clients need access to the database data, the highly available IP address must be available on the node, although it is not necessary to strongly enforce the ordering of the resource availability vis-à-vis either DB2 or the disk resource.

2.1 Disk Storage

DB2 can utilize the following for data storage:

- Raw disk (for example, direct use of /dev/sda with no mounted file system)
- Logical Volume Manager (LVM) managed logical volume
- File system (hosted by either a raw disk or a logical volume)

DB2 data can be stored either entirely on raw disk(s) (resulting in no file system dependency whatsoever), entirely on logical volume(s), entirely on file system(s), or a mixture of all three.

2.1.1 Raw disk

For high availability, the use of raw disk for data storage is the most straightforward approach. The identical path to raw disk (for example, /dev/sdb) appears across all nodes in the cluster that are physically connected to the storage. And, this same path is always available across all nodes that are physically attached to the storage, thus removing the need for any start or stop methods to make the disk visible to the host. A monitor method is likewise unnecessary.

Note that the true benefit of raw disk is that a possibly time-consuming **fsck** is never required, even in scenarios where the machine is halted abruptly.

2.1.2 Logical Volume Manager (LVM)

There are currently several logical volume managers available for Linux. We will discuss the LVM currently available as part of most Linux 2.4 based distributions. Note that it is important to remember that currently, LVM is not cluster aware. The implication is that, as root, it is very easy to cause an unrecoverable data loss in cluster environments.
In shared disk environments (the most common being shared SCSI for small two node clusters and shared fiber channel for larger clusters) more than one machine has physical access to a set of disks.
It is critical that LVM be shut down on all nodes other than the node from which any LVM administration is done. It is also critical that at most only one node have a file system mounted on a given logical volume (at any given time).

2.1.3 File system

A file system hosted by raw disk would require a resource to mount it prior to use, unmount it when it is no longer required, and monitor it to ensure file system health. This support is written as a resource of the class *IBM.Application*, and can be added to the resource group containing the DB2 resource, and DB2's IP address (collocated and with appropriate depends on relationship setup).
A file system on top of LVM managed logical volumes is handled identically.

2.2 Disk Protection

From the point of view of DB2, the disk protection is sufficient if the data on the raw disk, file system or logical volume is accessible for read and write operations. The DB2 engine itself ensures that only one instance accesses a database at any one time, as follows:

DB2 data is stored in logical entities called *tablespaces*. A tablespace is composed of one or more tablespace containers. A tablespace container is either a file, file system mount point, or a raw disk partition. DB2 does extensive checking when these containers are created and accessed. In particular, it is impossible to have another instance or database concurrently access the same raw disk, file system or file system mount point.
Disk protection (in the sense of protecting a file system from being mounted concurrently from two different physical hosts) is provided, if used, at the logical volume manager layer, but only for cluster-aware LVMs. If raw disk storage is chosen, care must be taken that the system administrator does not inadvertently concurrently mount the raw disk across multiple hosts. All file system mounting and unmounting tasks (for shared file systems) must be controlled by TSA.

2.3 DB2 requirements for the HA IP address

DB2 has no special requirements for the IP address. That is to say, it is not necessary to define a highly available IP address in order for the instance to be considered highly available. However, it is important to remember that the IP address that is protected (if any) is the client's access point to the data, and as such must be well known by all clients. In practice, this implies that this IP address is the one used by the clients in their CATALOG TCPIP NODE commands.
Note the following (which is not specific for HA IP addresses):
the service name specified by the database configuration parameter SVCENAME must be defined in the /etc/hosts files for all potential hosts
the DB2COMM variable must be set to TCPIP

3. Tivoli System Automation for Linux

TSA is a product that provides high availability by automating resources such as processes, applications, IP addresses and others in Linux-based clusters. To automate an IT resource (for example an IP address), the resource must be defined to TSA. Furthermore, these resources must all be contained in at least one resource group. If these resources are always required to be hosted on the same machine, they should all be placed in the same resource group. Further information on TSA can be found in the *IBM Tivoli System Automation for Linux Guide and Reference* manual.

Every application needs to be defined as a resource in order to be managed and automated with System Automation. Application resources are usually defined in the generic resource class *IBM.Application*. In this resource class there are several attributes that define a resource, but at least three of them are application-specific:

StartCommand
StopCommand
MonitorCommand

These commands may be scripts or binary executables.

There are some requirements for these commands:
You must ensure that the scripts are well tested, and will produce the desired effects within a reasonable period of time. This is necessary since these commands are the only interface between TSA and the application.

The monitoring script has to be efficient, as it is running at the frequency (in seconds) set in MonitorCommandPeriod and therefore can consume a lot of system time if written inefficiently.

The StartCommand and StopCommand must have a return code of zero for successful completion, otherwise an error is assumed and the resource is set to ‘Failed Offline’.

3.1 The DB2 for TSA Agent

The agent consists of the set of scripts necessary for the control of DB2 instances, DAS instances, and file system mount points, as well as utilities which simplify management of the cluster. Ensure that the agent is included with your version of DB2. To do so, ensure that the /opt/IBM/db2/V8.1/ha/salinux directory exists. If this directory does not exist (or if you are using an earlier version of DB2), perform the following procedure to install the TSA agent for DB2 and repeat these steps on each node in the cluster:

To install the agent:

Obtain the salinux.tar.gz package from **ibm.com**/db2/linux/papers, and then:

- cd $INSTALLPATH (eg. cd /opt/IBM/db2/V8.1)
- mkdir -p ha
- cd ha
- tar xvfz salinux.tar.gz

4. Tivoli System Automation for Linux and DB2 setup

This section describes the detailed setup of DB2 and TSA. The instructions are based on the following scenario, illustrated below in Figure 1:

- A two node cluster on nodes **node1 and node2**
- HA FastT storage connected to each node and containing the data disks
- One DB2 instance, **db2inst1,** running DB2 Version 8
- Home directory of the instance is /home_db2/db2inst1
- Instance data and database data are located on a file system on the disk **/dev/sdb** (a LUN provided by the FastT storage)

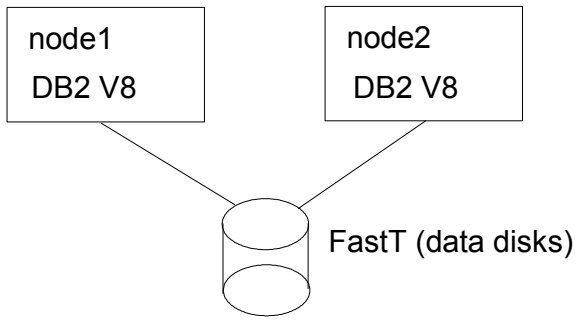

Figure 1: Simple two node cluster

4.1 Setup Overview

The setup consists of the following steps:
0. Set up the hardware
1. Install DB2
2. Create a DB2 instance
3. Install TSA
4. Setup TSA to manage the DB2 instance

4.2 Hardware setup

If redundancy is required, it is important to ensure that you have adequate hardware resources to provide redundancy. Please consult the IBM Tivoli System Automation for Linux Guide and Reference for detailed and current platform requirements.

4.3 Install DB2

DB2 has to be installed on all nodes that may potentially host a DB2 or DAS instance. Follow the *Quick Beginnings for DB2 Servers* manual. For Version 7 instances please follow the instructions given in the *Quick Beginnings for Linux* guide.
When installation is complete, your DB2 software will be installed in the /usr/IBM/db2/V7.1 directory (for Version 7) or /opt/IBM/db2/V8.1 (for Version 8).
Do not create instances at this step, that task will be deferred to the next subsequent step.

4.4 Create a DB2 instance

First, ensure that appropriate user and group IDs are created on each node of the cluster, and ensure that they are identical across each node. On each physical node participating in the cluster, three separate groups and user accounts need to be created for the:

- DB2 instance owner
- user who will execute fenced UDFs (user defined functions)
- DB2 administration server

If NIS or NIS+ is in use, groups and users must be created on the NIS server. If you are using local server authentication, repeat the following steps on all nodes in the cluster.
For example, use the following commands to create groups on each node in a local server authentication environment:

```
# groupadd -g 999 db2iadm1
# groupadd -g 998 db2fadm1
# groupadd -g 997 db2asgrp
```

Once the appropriate group IDs have been created, you must ensure there is a line in the /etc/fstab file similar to the following (for each node in the cluster):

```
/dev/sdb  /home_db2 ext2    noauto         1 1
```

Ensure that a /home_db2 mount point exists, so that the instance home directory can be mounted. To mount the home directory, execute the following command:

```
# mount /home_db2
```

You are then ready to create the user IDs as follows:

```
# useradd -g db2iadm1 -u 1004 \
    –d /home_db2/db2inst1 -m db2inst1
# useradd -g db2fadm1 -u 1003 \
    –d /home_db2/db2fenc1 -m db2fenc1
# useradd -g db2asgrp -u 1002 \
    –d /home_db2/db2as -m db2as
```

At this point, the appropriate user and group ID's exist at all nodes in the cluster. Ensure that the instance home directory is mounted at the primary node in the cluster and create the required instance. For example, to create a DB2 instance named db2inst1 for DB2 version 8, execute the following commands:

```
# mount /home_db2
# cd /opt/IBM/db2/V8.1/instance
# db2icrt -u db2fenc1 db2inst1
```

4.4.1 Create a DB2 instance: Restrictions

DB2 ESE multi-partition instances are not currently supported with the supplied DB2 for TSA package.

4.5 Install Tivoli System Automation for Linux

TSA must be installed on both machines locally. To install TSA follow the installation procedure described in *IBM System Automation for Linux on xSeries and zSeries, Guide and Reference*.

4.6 Set up Tivoli System Automation for Linux to manage the DB2 instance

Before proceeding further, please ensure that the cluster software is installed correctly, and ensure that the instance you wish to make HA is created, as described earlier in this white paper.

4.6.1 Step by step instructions

First, all nodes that will comprise the cluster must be prepared. The following command must be executed on every node. In this example the nodes are halin1 and halin2:

```
# preprpnode halin1 halin2
```

Create the TSA cluster domain:

```
# mkrpdomain halin_12 halin1 halin2
```

As you can see, the domain we have created is named halin_12, and consists of the nodes halin1 and halin2.
Start (online) the domain:

```
# startrpdomain halin_12
```

Ensure the domain is online:

lsrpdomain

You should see output similar to the following:

Name OpState RSCTActiveVersion MixedVersions TSPort GSPort
halin_12 Online 2.3.0.11 No 12347 12348

Ensure all nodes in the domain are online:

lsrpnode

You should see output similar to the following:

Name	OpState	RSCTVersion
halin2	Online	2.3.0.11
halin1	Online	2.3.0.11

For this two-node cluster, a reservation disk, or tie-breaker disk, is required (any two-node cluster will require exactly one tie-breaker). The tie-breaker disk is hosted exclusively by a shared disk (in other words, a disk that is not used to support any file system or data storage).

SCSI devices can be identified with four integers: each representing the host, the channel, the SCSI id, and the LUN. Note that these numbers may not be consistent across nodes, even if the target device to which they refer are the same physical disk.

For example, if a SCSI device is connected to two nodes (for example, halin1 and halin2) and has the following SCSI identifiers:

halin1: ID=0 LUN=4 HOST=2 CHAN=0
halin2: ID=0 LUN=4 HOST=2 CHAN=0

The characteristics of a SCSI device can be obtained with the dmesg command, for example:

```
# dmesg | grep “SCSI disk”
```

The output of the dmesg command will look similar like this

```
Attached SCSI disk sda at scsi0, channel 0,id 0, LUN 0
Attached SCSI disk sdb at scsi2, channel 0,id 0, LUN 0
Attached SCSI disk sdc at scsi2, channel 0,id 0, LUN 1
Attached SCSI disk sdd at scsi2, channel 0,id 0, LUN 2
Attached SCSI disk sde at scsi2, channel 0,id 0, LUN 3
Attached SCSI disk sdf  at scsi2, channel 0,id 0, LUN 4
```

If the disk ‘sdf’ is chosen to be the TieBreaker disk, then you can create the tie-breaker object as follows:

```
# mkrsrc IBM.TieBreaker Name=”scsi2” Type=”SCSI” \
DeviceInfo=”HOST=2 CHAN=0 ID=0 LUN=4” \
HeartbeatPeriod=2
# chrsrc -c IBM.PeerNode \ opQuorumTieBreaker=”scsi2”
```

Now the cluster is ready to host a highly available DB2 instance. Ensure that you have unprompted rsh access (as root) to the other nodes in this cluster. Unprompted root rsh access to all nodes in the cluster is not required once the instance has been made HA.
We will work with an instance named db2inst1 in this example. Ensure the instance is set up correctly. For example, if you are using local passwords, issue the following commands:

```
# rsh halin1 “cat /etc/passwd | grep db2inst1”
# rsh halin2 “cat /etc/passwd | grep db2inst1”
```

You should see output at each node similar to the following:

```
db2inst1:x:3957:3957::/home_db2/db2inst1:/bin/ksh
```

Ensure that the instance home directory is available. Notice that for this instance, the instance home directory will be mounted under /home_db2. Issue the following commands at each node in the cluster:

```
# rsh halin1 “cat /etc/fstab | grep home_db2”
# rsh halin2 “cat /etc/fstab | grep home_db2”
```

You should see output similar to the following at each node:

```
/dev/sdb        /home_db2       ext2    noauto    1 1
```

Ensure that the file system is not currently mounted at any node in the cluster. Issue the following commands:

```
# rsh halin1 “mount | grep home_db2”
# rsh halin2 “mount | grep home_db2”
```

You should see no output from these commands. If you discover that these file systems are mounted, please issue the appropriate **umount** command before proceeding.

Ensure that the /etc/services file is set up correctly (and the relevant entries are identical) across both nodes. Issue the following commands to verify this:

```
# rsh halin1 “cat /etc/services | grep db2inst1”
# rsh halin2 “cat /etc/services | grep db2inst1”
```

Ensure that the IP address that you wish to associate with this instance is not currently used (and consequently, available for use). We use the IP address 9.26.96.56 in the examples that follow.

In the next step, we create a highly-available DB2 instance named ‘db2inst1’, protect its home directory, /home_db2, and protect its well-known IP address 9.26.96.56. To do this, issue the following commands:

```
# cd /opt/IBM/db2/V8.1/ha/salinux
# ./regdb2salin -a db2inst1 -m /home_db2 –i \  -9.26.96.56
```

This may take a few moments, and the result will be a highly available instance. To verify this, issue the following command:

```
# ./getstatus
```

You should see output similar to the following:

```
      -- Resource Groups and Resources --


 Group Name            Resources
 ----------            ---------
 db2_db2inst1_0-rg     db2_db2inst1_0-rs
 db2_db2inst1_0-rg     db2_db2inst1_0-rs_mount
 db2_db2inst1_0-rg     db2_db2inst1_0-rs_ip
```

```
 -                    -


      -- Resources --

Resource Name            Node Name           State
-------------            ---------           -----
db2_db2inst1_0-rs           halin1          Online
db2_db2inst1_0-rs           halin2         Offline
                 -                -               -
db2_db2inst1_0-rs_mount     halin1          Online
db2_db2inst1_0-rs_mount     halin2         Offline
                 -                -               -
db2_db2inst1_0-rs_ip        halin1          Online
db2_db2inst1_0-rs_ip        halin2         Offline
                 -                                          -
-
```

This indicates that the instance and all of its associated resources are currently online on the cluster node halin1. The instance, its home directory, and its IP address are protected and ready for use. Any databases that are created and owned by this instance are also highly available, provided that all components of the database (all tablespaces, containers and log directories) are created within the /home_db2 mount point.

Now, let us assume that the instance is no longer required. Remove this instance from the cluster. To do this, issue the following sequence of commands:

```
# cd /opt/IBM/db2/V8.1/ha/salinux
# ./unregdb2salin -a db2inst1
```

Verify that there are no resources or resource groups remaining, by issuing the getstatus command.
Note that in general, for more complex database environments, multiple mount points will need to be associated with the db2inst1 instance. It is important to add all mount points being used by all databases controlled by these instances as resources, to the resource group containing the highly available instance.

5. Automate multiple DB2 instances with Tivoli System Automation for Linux

The previous chapter described the setup to manage a single DB2 instance with TSA This chapter discusses more complex scenarios with multiple DB2 instances. Since all of the following complex scenarios are built on the simple scenario from the last chapter, it is important to read and understand the previous chapter before going ahead.

Tivoli System Automation for Linux can manage DB2 in a variety of setups, for example:

2-node-cluster running 2 DB2 instances
n-node-cluster running m DB2 instances (2<=n<=8; m<=n)

Essentially, the only theoretical limitations to the cluster topology that can be supported are limitations of disk space or IP address resources. As a rule of thumb, if a particular node can locally host the required disk(s) containing the database data and executable code, and can host the IP address such that the clients have access to that IP address, then that node can host the DB2 resource group.

Note that at this time Tivoli System Automation for Linux supports up to eight physical nodes within a cluster.

In practice it is generally desirable to balance the cluster load, such that each node has a reasonable share of the set of instances hosted.
To automate multiple DB2 instances with Tivoli System Automation for Linux, one resource group has to be created for each DB2 instance as described in the previous chapter. TSA allows this resource group is allowed to run on more than one node. To ensure that only one resource group is running on a node at a time, a new relationship between the resource groups must be created. The relationship is *AntiAffinity*, which ensures that the resource groups are running on different nodes, if possible. Again, the goal here would be to attempt to balance the load across all nodes in the cluster.
On a four-node cluster running four DB2 instances, no more than one DB2 instance is running on a node at a time during normal operation. But if a node failure occurs, the DB2 resource group that was running on the failing node is restarted on one of the three remaining nodes.

Since the relationships in TSA have a direction, an *AntiAffinity* relationship must be created for each DB2 resource group, where the target resources are all other DB2 resource groups. The commands for two DB2 resource groups *db2rg1* and *db2rg2* are the following: (it is assumed that each of the two resource groups has been made HA as described earlier in this paper):

```
mkrel -p AntiAffinity -S IBM.ResourceGroup:db2rg1 \
        -G IBM.ResourceGroup:db2rg2 \ db2rg1db2rg2aarel

mkrel -p AntiAffinity -S IBM.ResourceGroup:db2rg2 \
        -G IBM.ResourceGroup:db2rg1 \db2rg2db2rg1aarel
```

If for any reason only one DB2 resource group is allowed to run on a node at a time, the relationship to be created is *AntiCollocated* instead of *AntiAffinity*. This relationship ensures that only one resource group is running on a particular node at a time. In this case there have to be more nodes in the cluster than the total number of DB2 resource groups (which is equivalent to the number of DB2 instances); otherwise, a failover of a DB2 resource group is not possible.

Finally, a resource group *db2rgall* can be created, which contains all the DB2 resource groups as members. Now all DB2 resource groups can be started and stopped by starting or stopping the *db2rgall* resource group. Note that the resource group *db2rgall* must be created with collocation *none*, which allows the members of the resource group to run on different nodes, otherwise the resource group will not start.

Furthermore the DB2 resource groups *db2rg1* and *db2rg2* must be added to the *db2rgall* resource group as *non-mandatory* members, otherwise all DB2 resource groups are stopped, if only one is failing. This is done with the following command sequence (Note that only ‘Offline’ resources can be added to a resource group so the DB2 resource groups *db2rg1* and *db2rg2* are stopped immediately, if they are running at the time when they are added to the *db2rgall* resource group.):

```
mkrg -l None db2rgall
addrgmbr -m F -g db2rgall IBM.ResourceGroup:db2rg1
addrgmbr -m F -g db2rgall IBM.ResourceGroup:db2rg2
```

6. Testing Highly Available DB2 instances that are automated with Tivoli System Automation for Linux

Implementing a High Availability cluster is a four-stage process. The stages are as follows:
1. Carefully plan the cluster.
2. Implement the cluster.
3. Test the cluster.
4. If the test results are not satisfactory, return to Step 1.

Planning the proper setup of a HA cluster is not within the scope of this document. Please see *IBM System Automation for Linux Guide and Reference* for advice on the setup of the cluster. We are providing some testing guidelines that are relevant for the use of DB2 in an HA environment.
Ideally every critical point of failure (in other words, a failure that has the potential to affect production operation) should be tested. 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 actually physically tested, it will never be entirely certain that the cluster will behave in practice as it does in theory. We will concentrate on failures internal to the cluster for the rest of this discussion.
The best testing environment is the High Availability 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, because this is one of the potential points of failure. Ensure that the failover takes place properly, and that the application is available and properly configured after failover.
If possible, also test the cluster with the expected workload; make sure that the failover setup is capable of handling the expected levels of activity.
When testing the availability of the application, be sure that accounts and passwords work as expected, that hostnames and IP addresses work as expected, that the data is complete and up to date, and that the changeover is essentially transparent to the user.

Configure a remote machine to be able to connect to the highly available DB2 database. A script can be easily written that will connect to the 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 failover is expected to take. Here is a brief sample script that may be useful for testing a High Availability cluster.

```
while :
do
db2 connect to database
db2 “select count(*) from syscat.tables”
db2 connect reset
sleep 60
done
```

A useful tool for checking the status of the SA Linux cluster is **getstatus** (found in the path of the DB2 for TSA package, usually /opt/IBM/db2/V8.1/ha/salinux). It can be used to report on the status of cluster resource groups and resources. An example output follows:

```
-- Resource Groups and Resources --


    Group Name         Resources
    ----------         ---------
db2_db2inst1_0-rg    db2_db2inst1_0-rs
db2_db2inst1_0-rg    db2_db2inst1_0-rs_mount
db2_db2inst1_0-rg    db2_db2inst1_0-rs_ip
           -                  -


-- Resources --


  Resource Name       Node Name          State
  -------------       ---------          -----
db2_db2inst1_0-rs        halin1         Online
db2_db2inst1_0-rs        halin2         Offline
           -             -              -
db2_db2inst1_0-rs_mount    halin1      Online
```

```
db2_db2inst1_0-rs_mount    halin2      Offline
            -              -             -
db2_db2inst1_0-rs_ip       halin1      Online
db2_db2inst1_0-rs_ip       halin2      Offline
```

The second indispensable tool is a short script to provide a check that DB2 is up and running. Catalog the database on the client system, and at each point during the testing where the cluster ought to be available, run the script and make sure that the system responds as expected. A sample script follows:

```
db2 -v connect to dbname
db2 -v “select * from syscat.tables”
db2 -v “select count(*) from usertable1”
db2 -v connect reset
```

This can be modified to test for the cluster’s own special characteristics. Points of failure that should be considered and tested are:

1. Correct installation of the software
2. Correct hardware configuration
3. Power failures
4. Network failures
5. Hardware failures in the CPU, the disk storage, and any of the physical infrastructures
6. Careless operator behavior
7. Software failures

Needless to say, this is not an exhaustive list. Cluster administrators are in the best position to know where points of failure are in their own clusters. An excellent procedure to follow for all tests is to log on as root on both nodes and execute on the server consoles the following command:

```
tail -f /var/log/messages
```

This will provide a continuously flowing stream of information about the status of the HA. Information that scrolls off the top of the screen can be retrieved by editing /var/log/messages.

It is worth keeping in mind that testing failures should be done in both directions (i.e. from primary to secondary and then back to primary).

1. Correct installation of the software and system

Following the procedures given in this white paper are sufficient to ensure that the system and software are configured correctly. However, the registration script **regdb2salin** which creates the appropriate resources and resource groups, is an important verification of this. Thus, while it is possible to bypass this step for creating DB2 resources and groups (and perform the work manually), it is strongly recommended for both cluster verification and for consistency that this script (and its inverse **unregdb2salin**) be the sole interface to creating and removing DB2 resources from an SA Linux cluster.

2. Correct hardware configuration

There are a number of tests to ensure that the hardware is correctly configured:

- use the **dmesg** command to check for any initialization errors
- 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

Testing 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. This will simulate the physical failure of individual network controllers, or of 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 command **ps -ef | grep inetd** will provide a list of running processes, including their group and PID. Selecting the inetd process and issuing a **kill -9 <PID>** command will simulate the failure of the network software.

5. Hardware failures

Hardware failures can be simulated by a variety of brute force methods. CPU failures can be mimicked by hitting power or reset switches, by killing processes, or by 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
You may also make liberal use of the halt, power off, and reboot commands for this aspect of testing.
A good way to test for hardware failures in the drive array is to pull out power cables to the array, or pull out the fiber or SCSI connections to the host.
6. Careless operator behavior
This is difficult to test, since generally this sort of HA is most useful for protecting against single points of hardware or software failures. Protecting against careless operator behavior (for example, an errant transaction) is outside the scope of this document.
7. Software failures
Software failures can be simulated by killing critical processes, which also works to simulate many operator errors. You can find the Process ID for the *rscd* processes using **ps -ef** | **grep rscd**. The processes can then be killed using a **kill -9 <PID>** command. 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.

IBM Canada
8200 Warden Avenue
Markham, ON
L6G 1C7
Canada