DB2 UDB V8.1 ESE and high availability on AIX with HACMP 4.4.1

A mutual takeover scenario

Mika Nikolopoulou July 23, 2004

High availability of your data is a must for the on-demand environment. This article takes you through a specific example of how the author set up high availability as part of a proof-of-concept demonstration of IBM® DB2® Universal Database™ with High Availability Cluster Multi-Processing (HACMP) on IBM AIX®.

Introduction

High availability of data is vital for on-demand solutions to your business problems. If you're a regular developerWorks reader, you may already be familiar with the solutions that are available for providing high availability, discussed in articles such as DB2 Universal Database and the highly available data store. That article will take you through the reasons for high availability and discuss your options.

But if you're closer to the implementation phase, you might find it valuable to look at a specific example of how a high availability and disaster recovery scenario for DB2 UDB Enterprise Server Edition (ESE) on AIX was set up at a customer site.

The application was a system used by technical service employees who were dispatching technical problems to the technical resolution units throughout the United States. It was an online transaction processing (OLTP) application with 10,000 users accessing the system in order to dispatch technical problems.

There was a special need for high availability for this system. This system was a proof-of-concept (POC) system implemented in near-real production conditions. During this POC we tested DB2 UDB V8.1 ESE with high availability and replication. Replication was used for disaster recovery purposes, using peer-to-peer replication with update conflict detection. This article focuses on the high availability aspects of the POC and will not cover the replication part.

With this POC, we also tested expansion and scalability capabilities of DB2 UDB's architecture, because the project was in the process of expanding to more clusters in other geographic areas.

Today, the system deploys only one cluster with two servers in the New York area. In the future there will be an additional cluster in Maryland and a standalone server in Florida to provide additional availability and disaster recovery capability. In our POC we simulated the existence of all three environments and sets of servers (one cluster in New York, and two standalone servers in Florida and Maryland), with the only difference being that they were all physically located in the customer lab in New York for testing purposes.

We wanted to test for performance, reliability, stability and the requirement to achieve the fastest failover time with a target failover time below 60 seconds. We set up a full high availability mutual takeover scenario and a three-way peer-to-peer replication with conflict update setup in less than a week. This was implemented on a total of four servers, two of them in a cluster. Each server had two instances of DB2, since we were using DB2 UDB ESE without the partitioning feature and we needed to implement mutual takeover.

We chose mutual takeover rather than a standby configuration for two reasons:

- The customer wanted maximum availability. This means that while one server is down, the other is running uninterrupted. So half of the application requests, directed to one server, do not experience any problem, and the other half of the requests, directed to the failing server, will have to migrate to the failover server. This is not possible in a standby scenario, because all client/application requests are directed to a single server and they will all fail in the case of a failure of the primary server. The cost is that both machines must be configured to handle double the workload, although under normal operation this capacity is used only by half. Also, a mutual takeover can only be implemented if the two applications are distinct and use different databases.
- The customer wanted to use both servers and not have any server idle at any time. In a standby configuration, the standby server will be idle while it waits for the production server to fail. This is an important argument, although we can respond that in a mutual takeover environment the two servers are running at half their capacity, so under normal circumstances, we are not fully exploiting them, either.

The first factor was a very strong driving force behind our decision to implement mutual takeover with this particular project, especially because the customer was running two applications and two databases.

How DB2 UDB works in a high availability environment

The software used to implement clustering and failover processes on AIX is IBM HACMP. HACMP does the following:

- It continuously sends the heartbeats between the servers in the cluster to identify if and when any of them are down.
- It provides tools to define and manage the group of resources that participate in the high availability setup, including shared disks, network adapters, and application servers that will handle the starting and stopping of the database.
- It handles the failing over of the resource groups from one system to the other as needed after a failure or after a coming back of a server from a failure.

DB2 UDB comes with a set of scripts that define its behavior in case a failure or a recovery of a system takes place. In a DB2 UDB single-partitioned environment, the script performs the following actions:

1. If one server fails, it will start DB2 UDB on the standby server.
2. If a failed server reintegrates into the HA cluster (that is, becomes available again), it will stop the database on the standby server and start it on the reintegrated server. This also depends on the HA cluster configuration.

The basic script is called `db2.pe`. It comes in two versions:

- `db2.pe.ee` - for a database with a single partition. This script is very simple, containing only the starting or stopping of the database.
- `db2.pe.eee` - for the multi-partition environment. This script is more complex because it starts multiple partitions, cleans the system, and updates some configuration parameters and files for the proper failover of each partition.

In reality this script (which starts the database or partition on another server when failover occurs, or stops the database on the standby server and starts it back to the original one when a fail back occurs) is the most important task that DB2 UDB performs in a high availability environment. This is because of its shared-nothing architecture. (For more information on shared-nothing architecture, see the explanation in the DB2 Information Center. In a non-concurrent access environment, where the disks are simultaneously accessed by both servers, the delay in a DB2 failover is mostly due to the need to varyon and varyoff the shared volume groups (HACMP terminology), or in other words, to move control and access of the shared disks and resources from one server to another. This task of transferring control of access to the shared disks from one server to the other must take place in both the partitioned and non-partitioned environments.

This task alone -- moving access to the shared disks from one server to another -- can take significant time, depending on how big the disks are and how many there are. In our case, because we were using a single disk per volume group and two volume groups, and the disk space was not high (only 9GB per disk, for a total of 18GB in two disks), the failover time for disks was about 20 seconds. This time can be significantly reduced if you use raw devices and if you use concurrent access.

We used system managed space (SMS) and journaled file system (JFS). JFS is required for SMS on AIX, and no concurrent access is possible in this case. If we had to add more disks, this would increase the failover time. By using raw devices this overhead is reduced. To further improve failover performance, we could use raw devices, especially if we have many disks and not just one or two as in our example. With database managed space (DMS) and raw device, we can decrease the time to fail over the disk resources, because there will be no checking of disks during failover. We can also have the **concurrent access on** for the raw disks, and that even further reduces failover time. HACMP will still have to fail over all other resources (like the network adapters), but the disk failover, which is the longest, will be minimized. Therefore, you should use concurrent access for faster failover time.

In HACMP version 5.1 this capability is implemented in a simple and elegant way by using the option of enhanced concurrent access when setting up the shared disk access. This feature eliminates the need to switch access to the disk from one server to the other. At the time of our setup, HACMP 5.1 was not available.

Customer environment

The clusters cover the need to implement high availability with the highest possible availability time. The customer wanted to keep the clusters in different locations for load balancing and disaster recovery. Replication is used in this case for disaster recovery. The database is relatively small, only 5-10 GB in size at this point. The application is a C application, using WebSphere® as a Web application server.

The goal for the POC was to implement high availability with a target of achieving higher than 99.999% availability.

Installation and configuration

The tasks to set up a high availability environment with HACMP on AIX are:

1. Define the volume groups and the logical volumes
2. Define the adapters
3. Install DB2 UDB
4. Install and customize the DB2 HA scripts
5. Define application servers
6. Synchronize the cluster
7. Tune DB2 for performance
8. Customize the application to reconnect in case a *connection* fail error message is returned
9. Test

POC environment

Hardware configuration

The configuration includes two RISC/6000 servers 7026H70 with four CPUs each, 1GB memory, running AIX 5.1 maintenance level 3. This was a cluster with HACMP 4.4.1 installed (initially it was 4.4.0 but this level is not supported with AIX level 5L). The two servers were connected to a SSA shared disk array 7133-020 with RAID 5 mirroring and a total of 136.1GB of storage. Although there were 15 disks, we could only see five disks, each 9.1GB. For this test we used only two disks, one disk per instance. This was the cluster of two servers. The names of the servers were *HADB1* and *HADB2*.

Software configuration

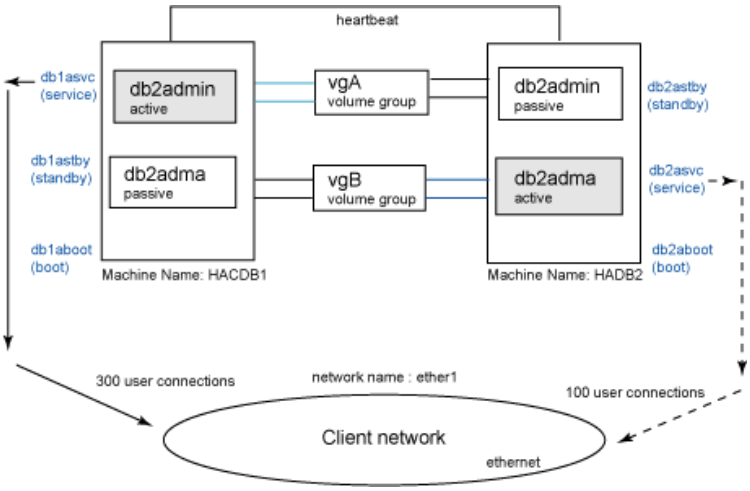
The customer wanted to have an active-active or mutual-takeover implementation. The customer wanted to keep two separate instances in each environment in order to isolate their systems since they were running different applications with two different databases.

The requirement was to keep two separate environments for the two different applications, including two different databases. The reason they wanted to separate the instances and have

both servers running was to decrease the possibility of multiple concurrent application failures. With separated environments, if one server goes down, the other instance on the second server can still be up and connected. If both applications are running on one server in a hot standby configuration, then if that server fails, both applications and both sets of users which are accessing the applications would have to failover. Now, with two different servers, instances, and databases, a set of users can always be active even if one server fails. If both servers fail, the applications are directed to the replica nodes using replication to a third disaster recovery server set.

We therefore decided to have a mutual takeover scenario, with DB2 UDB ESE V8.1, and two instances per server. Each instance would be active on one server and passive on the other server. Figure 1 shows a diagram of a normal operation:

Figure 1. Normal mutual takeover scenario



From this figure you can see that each server has three adapters. (Actually there are four adapters. We do not show the fourth because the fourth adapter is the back door adapter for the use of the administrator only.)

- The first is the boot adapter with a distinct or persistent IP address. This is where the machine boots.
- The service adapter has a floating IP address. This is the adapter where all applications connect to the primary instance for each server. The fact that the IP address of this adapter is floating means that no matter where the instance is running, on server HADB1 or on server HADB2, this floating IP address remains the same and the application does not see any change -- except of course a disconnect and a reconnect.
- The third adapter is the standby adapter. This adapter is passive during normal operation, and waits for a failure to occur. When a failure happens on one server, then this standby adapter takes over all connections from the other server. This standby adapter becomes the service adapter for the instance that is failing over. Therefore, this adapter becomes the physical address behind the floating address of the service adapter when a failure happens.

With this topology, we have a single point of failure. That point of failure would be in case we were in a failover mode, with both instances active on one server. If one of the two adapters fails at that

point, then there is no additional standby adapter to be used as a backup. To recover we would need to have a fourth adapter as a standby adapter. For the very rare case when both adapters fail we would need to have two additional standby adapters to cover all potential single point of failure cases.

The names en0, en1 and en2 correspond to the physical device name of the adapters. The names of the adapters as they are configured in the cluster topology and used by the applications are:

For server HADB1:

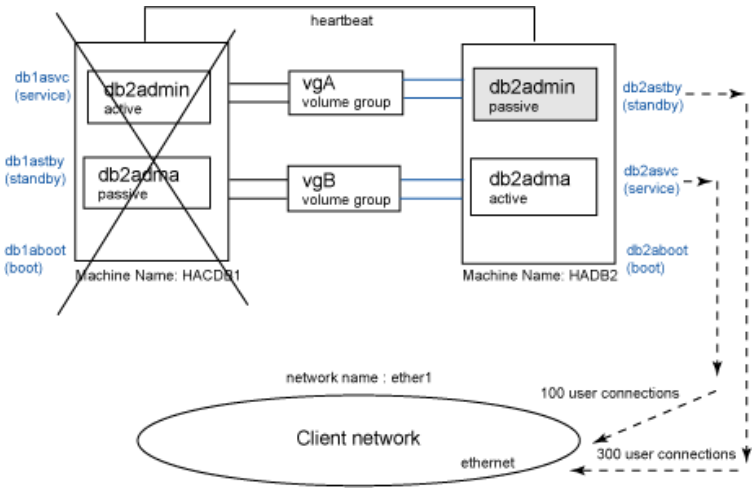
- En0 -> db1aboot, boot adapter, used for booting the machine. It always has a distinct address
- En1 -> db1asvc, service adapter with floating IP address 128.209.61.3
- En2 -> db1astby, standby adapter which can take over the service adapter floating IP address from server HADB2

For server **HADB2**:

- En0 -> db2aboot, boot adapter, used for booting the machine. It always has a distinct address
- En1 -> db2asvc, service adapter with floating IP address 128.209.61.4
- En2 -> db2astby, standby adapter which can take over the service adapter floating IP address from server HADB1

Figure 2 shows the status after a failure has occurred on server HADB1 and the application has been taken over by the second server, HADB2.

Figure 1. Mutual takeover after a failure



When a failure occurs, for example server HADB1 fails, then all users connected to HADB1 are disconnected from the database. What the application perceives is only a message that there is no connection to the database.

Applications should be aware of the error codes, listed in Table 1, which may be encountered during failover in an HA environment, and should attempt to reestablish the connection to continue:

Table 1. DB2 error codes which may occur during failover

SQLCODE	SQLSTATE
-900	08003
-1015	55025
-1034	58031
-1035	57019
-1224	55032
-1229	40504
-6036	n/a
-30081	08001

So when the server HACDB1 fails, the application should receive one of the messages in the table above. In this case, the application will enter a loop and try to reconnect until the failover is complete and the instance on server HADB1, which is the instance db2admin, is available again on the standby server HADB2.

When the failover is complete, the instance is available, and the connect statement in the application is successful. The application then exits this loop, and continues operation as normal.

This is what happens on the application side. You can find a very detailed description of the series of events that take place during failover, as well as their timing (the time each event contributed to the total failover) in the hacmp.out file in the HACMP directory. From the DB2 side, all DB2 activities can be tracked in the `db2diag.log` file under the `sqllib` directory of DB2 UDB.

Setting up HA

Step 1 - DASD setup for DB2 UDB

First we need to prepare the disks and volume groups that will be used in the configuration.

We need to define the volume groups. Based on our design, each instance will be assigned a volume group, which will be a shared disk capable of failing over from one server to the other upon failure of a server and back. We decided to not have automatic failing back to the original server. This means that when the original server is back at a normal state, the disk will not automatically switch back. This is because it is possible that when the server comes back up again, the administrator may want to perform maintenance work and does not want to have the instance immediately transfer back to the original server. So the automated reintegration of the server in the cluster is disabled for the first phase of testing.

Because we have two instances, we need two volume groups. Each instance needs its own volume group. For faster failover time the best configuration is to use DMS tablespaces and raw devices, and even better allow concurrent access to the disks. This means that during failover, the HACMP software does not have to varyoff and varyon the disk -- meaning transfer the connection from the failing server to the active server so that the shared disk is accessed by the active server because of the concurrent access. Raw device usage also improves failover time in case the

database is large, because during failover the operating system does not have to do a file check to the file system (fsck).

In our case, because the database is so small, only 1-2GB at this point, we simplified our design by using only the default SMS tablespace, and for this reason we can only use a JFS (journaled file system). The volume group will hold only a single disk, and in this disk, we defined the three shared instance directories, which are the home directory for the instance owner, the directory for the fenced id and the directory for the Administration server.

For the DB2 instance db2admin, we will use volume group **vgA** and for the DB2 instance db2adma the volume group **vgB**. Instance db2admin will have primary server HADB1 and db2adma will have primary server HADB2. Each will fail over to the other server as secondary. The user IDs for the two instances are as follows, and the definition has to be exactly the same in both servers. This includes even the definition of port numbers and service names for TCP/IP connections. All definitions and setup for DB2 UDB must be identical in the two servers. Of course the reason why each instance must have exactly the same configuration, userids, file systems, port numbers, and so on in both servers, is that the application should not see any difference between the two servers. If one server fails over to the second server, then the application will experience no problems due to difference in directory names and so on.

Here is our configuration:

System: HADB1

- administration server userid: radmin
- password: radmin
- 32-bit instance
- single partition

- Instance userid for first instance: db2admin
- password: db2admin
- service name: db2c_db2admin
- port number: 50000
- Fenced userid: db2fence
- Fenced password: db2fence

- Instance userid for second instance: db2adma
- service name: DB2_db2adma
- port number: 60004
- Fenced userid: db2fenca
- Fenced password: db2fenca

- Authentication: client
- Autostart off

We need to manually add two entries to the etc/group file where the SYSADM/ instance owner group ids are defined. We added the line:

- db2admg: db2admin, db2adma
- The path for installation is: /home/db2amdin,
- /home/db2fence
- /usr/opt/db2_v08_01 for binaries
- /home/db2amda,/home/db2fenca

System: HADB2

- administration server userid: radmin
- password: radmin
- 32-bit instance
- single partition
- Instance userid for first instance: db2admin
- password: db2admin
- service name: db2c_db2admin
- port number: 50000
- Fenced userid: db2fence
- Fenced password: db2fence
- Instance userid for second instance: db2adma
- service name: DB2_db2adma
- port number: 60004
- Fenced userid: db2fenca
- Fenced password: db2fenca
- Authentication: client
- Autostart off

We needed to manually add two entries to the etc/group file where the SYSADM/ instance owner group IDs are defined and we added the line:

- db2admg: db2admin, db2adma
- Path for installation: /home/db2admin,
- /home/db2fence
- /usr/opt/db2_v08_01 for binaries
- /home/db2amda,/home/db2fenca

We should configure HACMP BEFORE installing DB2 UDB, so that the volume groups and the disks are already in place for the DB2 UDB installation and configuration.

Therefore, the first step is to define the shared DASD in HACMP.

1. Define the disk drives. In our system, we only had to use one disk per instance. The disk has the physical name pdisk10 for the first instance, which will be assigned the volume groups **vgA,** and a logical name hdisk16 on servers HADB1 and HADB2. For **vgB** , which will be assigned to the second instance, the physical disk name is pdisk11 and the logical name is hdisk17 on the servers HADB1 and HADB2.

	First Instance (db2admin)	Second Instance (db2adma)
Volume group name	vgA	vgB
Physical disk name	pdisk10	pdisk11
Logical disk name	Hdisk16 for HADB1 and hdisk17 for HADB2	Hdisk16 for HADB1 and hdisk17 for HADB2

Server HADB1

PDISK	HDISK	Serial #	PVID	Volume
Group				
pdisk10	**hdisk16**	**294CF915**	**00015198ad4cd369**	**vgA**
pdisk11	**hdisk17**	**294D02DD**	**00011233737e8532**	**vgB**
pdisk12	hdisk8	29CC9F0A	none	None
pdisk13	hdisk9	29D27E33	none	None
pdisk14	hdisk10	29D291A8	none	None
	hdisk11	3E42AE12	0001156073b90800	None

Server HADB2

PDISK	HDISK	Serial #	PVID	Volume
Group				
pdisk10	**hdisk16**	**294CF915**	**00015198ad4cd369**	**vgA**
pdisk11	**hdisk17**	**294D02DD**	**00011233737e8532**	**vgB**
pdisk12	hdisk18	29CC9F0A	none	None
pdisk13	hdisk19	29D27E33	none	None
pdisk14	hdisk20	29D291A8	none	None
	hdisk6	3E42AE12	0001156073b90800	None

Use the `lsdev` command to get a list of all disks in your system. You should have the same number of disks on both servers for this configuration to work properly. In this case, we only used one disk, but if you are going to have more than one disk per server, make sure that you have the same amount of disks on each server. Read the White Paper DB2 Universal Database Enterprise Edition for AIX and HACMP/ES at DB2 Universal Database and the highly available data store, page 6, for more details about symmetry in disks and exact AIX commands.

2. After you have defined the disks, you can define the volume groups. The volume groups on AIX are groups of disks that will be used in the shared disk environment and they will be part of the set of objects that will be able to failover from one server to another and back as needed. These disks are collected in a group, called the *volume* group.

In our case, we defined everything on HADB1 and then synchronized the definition to the other server HADB2. We did not have to make the same definition twice. In our example, on HADB2 we defined two volume groups, vgA and vgB, with disks hdisk16 and hdisk6, respectively. This

definition was applied automatically to the HADB2. We then attach each volume group to its primary server.

For vgA, we attach it to HADB2 by using the command from HADB1:

```
# varyonvg vgA
```

And from HADB2 we use the command :

```
# varyonvg vgB
```

Now the two volume groups are available and attached to their respective servers as in normal operation.

3. Create the JFS log before creating the logical volumes for the volume groups. The reason for this is that the name of the JFS log must be unique. In our case, because we would only create a single JFS per volume group, we did not have to specifically name the JFS Log and we skipped this step. More details on page 7 of the White Paper “DB2 Universal Database Enterprise Edition for AIX and HACMP/ES".

4. Create the logical volumes and the JFS sytems. They have to have unique names and should not be currently defined on any node. Also the file systems should be set so that they are not mounted on restart.

In our case we created the logical volumes per volume group. Each logical volume corresponds to a directory.

Table 2. Volume group directories

Volume Group	Directory	Logical Name
vgA	/HOME/db2admin	admin
vgA	/HOME/db2fence	fence
vgA	/dbA	dbA
vgB	/HOME/db2adma	db2adma
vgB	/HOME/db2fenca	fencea
vgB	/dbB	dbB

Each logical name will create a new JFS. Having a different logical name for each disk (if raw devices) or file system (if SMS, and thus JFS) has its advantages and disadvantages. (**Note**: In AIX you can only use JFS for SMS tablespaces.)

The advantages of having multiple JFSs is the optimized performance with parallel I/O. This is more important in large databases. For faster failover performance, the fewer file systems, the better.

We are using RAID5, so we will not mirror the logical volumes.

Then mount the filesystems created because they are not automatically mounted. During the creation of the filesystems, we defined that they should not automatically be mounted. After changing the ownership of the filesystems to assign them to the instance owner userids, we unmount the filesystems and we varyoff the volume groups. Then we import the volume groups we created on the HADB1 server to the HADB2 server with the same major number. It should not be automatically activated on restart so that it is only activated when HACMP starts it.

Finally after both servers are set up, return to normal operation, which means that we need to attach the vgA (varyonvg vgA) to the HADB1 server and attach the vgB to the HADB2 issuing from HADB2 the varyonvg vgB.

With this, we have completed the setup of the disks for storage.

Installing and setting up DB2 UDB

1. On HADB1, initially create the userids for DB2 UDB and install DB2 using the **db2setup** command. In our case, after initial installation of DB2, we also installed an additional instance using the command **`db2icrt -u db2denca db2adma`**.

`db2fenca` is the fenced userid and `db2adma` is the second instance ID. We also updated the `tcpip` group file under the `etc` directory, in order to reflect the second instance. Also check the etc/services file in order to confirm that the port numbers are assigned properly.

After installation of DB2 UDB on HADB1, install the HACMP scripts using the `db2_inst_ha.local` script. This script is located at `/usr/opt/db2_08_01/sample/hacmp/es`. You can simply run it using the following command: **`#./db2_inst_ha.local db2admin.SAMPLE`** (where db2admin is our instance owner ID, "." stands for the directory where we are issuing the command from, and SAMPLE is the name of the database we want to use during failover.)

This will install the HA scripts to the `/usr/bin` directory. The only script that we will use in this configuration is the `rc.db2pe` script. Make sure that this is linked to the `rc.db2pe.ee` version, because there is also the `rc.db2pe.eee` version for multiple partitions.

This script is simply starting and stopping the database instance, based on the parameter that you give to the script when you issue it. The script is used by HACMP either as `rc.db2pe start` or as `rc.db2pe stop`. In the first case the script reads the start option and starts the instance, and in the second case, it simply stops the instance with the force option.

The HADB1 node is now set up from a DB2 UDB perspective, and you unmount the filesystems and varyoffvg vgA and vgB. Then on the HADB2 node varyonvg vgA and vgB and mount back all filesystems on the HADB2 server.

2. On HADB2, we need to delete the directories related to the instances we created on HADB1 because we will not be allowed to install the two instances on the second server since the shared disks on vgA and vgB contain these directories already. Therefore, we need to uninstall them on HADB2 and install them again.

Therefore, on HADB2, delete the directories `/home/db2admin/sqllib` and `/home/db2adma/sqllib` using the **`#rm -rf /home/db2admin/sqllib`** and **`#rm -rf /home/db2adma/sqllib`** commands.

- Then create the userids and install DB2 UDB using the **db2setup** command. Repeat the same steps as before for HADB1, creating both instances and installing the HACMP scripts.

After finishing setting up both instances and checking that you can start and stop the instances, then unmount all directories and on HADB2 issue both:

```
varyoffvg vgA
varyoffvg vgB
```

This will get us ready to complete our setup with the HACMP configuration.

We should always be careful to maintain the same level, software, IDs, and to set up like instance and database configurations on both servers moving forward, in order for HA to work properly. The same is true for system parameters like maxuproc, port numbers, and AIX maintenance levels.

On server HADB1, Db2nodes.cfg looks like one of the following:

```
0 floating IP address 0
0 128. 209.61.3 0
```

On server HADB2, Db2nodes.cfg looks like the following:

```
0 128. 209.61.4 0
```

Because the address in db2nodes.cfg is floating, there is no need to edit the file when failing over from one server to the other. For the two instances, we are using two application servers and two resource groups. We have one resource group per instance. Each resource group will contain a script to start and stop that instance. The file .rhosts contains a list of all IP addresses but with no userids. The file .rhosts should be used and hosts.equiv should not be used. The diagnostic file under error number `SQL6048N` is not giving the right information that hosts.equiv is used. It is NOT used, and .rhosts is used instead. For TCPIP, we had to edit the TCPIP group file under `etc` in order to define the second instance within that file.

Configuring HA for faster DB2 UDB failover

For faster failover times, these are the DB2 UDB configuration settings to tune:

db2empfa databasename

The db2empfa connects to the database partition in exclusive mode. In SMS it allocates empty pages to fill up the last extent of all index and data files that are larger than one extent. It changes the multipage_alloc configuration parameter to Yes and it disconnects.

The multipage_alloc is used for SMS tablespaces only, and it is used to improve insert performance. Once set to Yes, it cannot go back to No. It applies to all SMS tablespaces and cannot be applied to individual spaces. The reason for using the command above is to make sure that there is enough space available on SMS tablespaces, so that there will be no unnecessary additional delays to allocate pages after a failover.

softmax= 20 (for OLTP)

The softmax parameter is set to a lower number, so that the log control file is written to disk more often, and thus after a crash recovery the database may need less time to recover. This is because the log control file will be most updated and unnecessary writes to disk will be avoided. This is documented in the DB2 Performance Guide.

num_iocleaners= 20

More I/O cleaners means that in case of soft recovery, the contents of the database will be more updated on disk, because I/O cleaners clean bufferpools from contents and therefore this will reduce the recovery time.

logfilesz=1000 (default)Chngpgs_thresh=20 (30)

The lower the value, the lower the percentage of changed pages in the bufferpool that is required to trigger the asynchronous page cleaners to start cleaning the bufferpool. This improves performance. In HA this also means that more pages/data are written to disk more often and therefore less recovery time is needed in case of a crash recovery, since data committed are often written to disk.

Heartbeat rate

The heartbeat rate should not be below eight seconds. In reality the heartbeat rate is one second, minimum. The cycle value is four seconds, minimum. We multiply 4 by 2, which is the two minute interval to monitor the heartbeat to get the eight second minimum.

This becomes eight second intervals 2x4 (only monitor the heartbeat every two seconds) only if the load on the system is reasonably controlled. This means it should not be an overloaded OLTP application. For heavy transactional databases and for business intelligence applications, the heartbeat should be higher. Also, the network cannot be completely overloaded with packets; otherwise the heartbeat will be lost.

There are two primary reasons for keeping the heartbeat above the minimum. One reason is that the heartbeat competes with other resources in the system and the other reason is that in case there is any delay in getting back a response, and the heartbeat does not wait long enough, then failover may occur for no reason. The network cannot be completely overloaded with packets; otherwise the heartbeat will be lost. You should use a non-IP heartbeating, like RS232 or target mode SSA or target mode SCSI. Then failover will only occur if both heatbeats, RSCT and the non IP heartbeat fails. It is always good to use a dual heartbeat network.

Configuring HACMP

Often a database administrator is not familiar with HACMP terminology. So I'll take time here for a brief introduction, because HA in reality is performed by HACMP. DB2 only uses a script which is triggered by HACMP in order to start or stop the instances based on failover need.

HACMP terms

Resource group: This is a group of all resources that need to be failed over from one server to the other. In a resource group we define all resources that should be attached to the DB2 UDB server that we want to failover. These include the disks as defined in the volume groups, the applications, that we will define and they will constitute the DB2 HA scripts, the networks and the adapters.

Define the cluster ID and name. In our case it will be ID number **999** and name **db2cluster**. Then define the cluster nodes, and in our case they are **HADB1** and **HADB2**. Then we add the adapters. In our case we have four adapters per server and we only use three. This is described in the picture provided. One of the adapters is the boot adapter, not used for HACMP. Then the service adapter is the primary adapter per server for the primary instance on each server and has allocated a floating address. So for HADB1 where the primary instance is db2admin, the service adapter will host all connections to the instance db2admin and on HADB2 the service adapter will receive all connections to the instance db2adma. When for example HADB2 fails, then the service adapter on HADB1 will continue to get connections for db2admin while the standby adapter on HADB1 will get connections for the now active db2adma second instance on HADB1. In case there is no failover the standby adapters remain idle.

Here is the adapter topology:

Cluster Description of Cluster db2cluster
Cluster Description of Cluster db2cluster
Cluster ID: 999
Cluster Security Level Standard

There were two networks defined: ether1, snet1

There are two nodes in this cluster:

NODE HAdb1:

This node has two service interfaces:

Service Interface db1asvc:

- IP address:128.209.61.3
- Hardware Address:
- Network:ether1
- Attribute:public
- Aliased Address?:unknown

Service Interface db1asvc has one boot interface

- Boot (Alternate Service) Interface 1:db1aboot
- IP Address: 128.209.61.75

- Network: ether1
- Attribute: public

Service Interface db1asvc has one standby interface

- Standby Interface 1: db1astby
- IP Address: 128.209.67.54
- Network: ether1
- Attribute: public

Service Interface db1atty2:

- Hardware Address:
- Network: snet1
- Attribute: serial
- Aliased Address?: False

(INVALID) Service Interface db1atty2 has no boot interfaces

Service Interface db1atty2 has no standby interfaces

NODE HAdb2:

This node has two service interfaces:

Service Interface db2asvc:

- IP address: 128.209.61.4
- Hardware Address:
- Network: ether1
- Attribute: public
- Aliased Address?: unknown

Service Interface db2asvc has one boot interface

Boot (Alternate Service) Interface 1: db2aboot

- IP Address: 128.209.61.76
- Network: ether1
- Attribute: public

Service Interface db2asvc has one standby interface

Standby Interface 1: db2astby

- IP Address: 128.209.67.55
- Network: ether1
- Attribute: public

Service Interface db2atty2:

- IP address: /dev/tty2
- Hardware Address:
- Network: snet1
- Attribute: serial
- Aliased Address?: False

(INVALID) Service Interface db2atty2 has no boot interfaces

Service Interface db2atty2 has no standby interfaces

Breakdown of network connections:

Connections to network ether1

Node HAdb1 is connected to network ether1 by these interfaces:

- db1aboot
- db1asvc
- db1astby

Node HAdb2 is connected to network ether1 by these interfaces:

- db2aboot
- db2asvc
- db2astby

Connections to network snet1

Node HAdb1 is connected to network snet1 by these interfaces:

- db1atty2

Node HAdb2 is connected to network snet1 by these interfaces:

- db2atty2

Then synchronize the setup with the other node, HADB2.

At this point, we have completed the definition of all resources per server; we now need to define the resource group. We will add two resource groups, rgA and rgB.

For each resource group, we will add the primary and secondary servers. For resource group rgA, the primary server is HADB1 and it has to be placed first in the definition. For resource group rgB, the primary server is HADB2 and it has to be placed first in the participating node names definition upon definition of the new resource group.

In each resource group, we add an application server. The application server will take care of the starting and stoping of the database in various cases of failover scenarios. The application server wil be called app1 for the first instance and app2 for the second. For both servers, the start script

will be the `/usr/bin/rc.db2pe db2admin start` and the stop script will be the `/usr/bin/rc.db2pe db2admin stop`. The same applies to the second instance db2adma.

Here is the output:

```
Resource Group Name: rgA

Node Relationship: cascading

Participating Node Name(s); HAdb1 HAdb2

Node Priority

Service IP Label: db1asvc

Filesystems: /dbA /home/db2admin /home/db2fence
Filesystems Consistency Check: fsck

Filesystems Consistency Check: fsck

Filesystems/Directories to be exported

Filesystems to be NFS mounted

Network For NFS Mount

Volume Groups: vgA

Concurrent Volume Groups

Disks

AIX Connections Services

AIX Fast Connect Services

Shared Tape Resources

Application Servers: app1

Highly Available Communication Links

Miscellaneous Data

Automatically Import Volume Groups false

Inactive Takeover: false

Cascading Without Fallback: false

9333 Disk Fencing: false

SSA Disk Fencing: false

Filesystems mounted before IP configured: false
```

Run time parameters:

Node Name: **HAdb1**
Debug Level: high
Host uses NIS or Name Server: false
Format for hacmp.out: Standard

Node Name: HAdb2
Debug Level: high
Host uses NIS or Name Server: false
Format for hacmp.out: Standard
Resource Group Name: rgB
Node Relationship: cascading
Participating Node Name(s): HAdb2 HAdb1
Node Priority
Service IP Label: db2asvc
Filesystems: /dbB /home/db2adma /home/db2fenca
Filesystems Consistency Check: fsck
Filesystems Recovery Method: sequential
Filesystems/Directories to be exportedFilesystems Recovery Method: sequential
Filesystems to be NFS mounted Filesystems Recovery Method: sequential
Network For NFS MountFilesystems Recovery Method: sequential
Volume Groups: vgBFilesystems Recovery Method: sequential
Concurrent Volume Groups Filesystems Recovery Method: sequential
Disks Filesystems Recovery Method: sequential
AIX Connections ServicesFilesystems Recovery Method: sequential
AIX Fast Connect ServicesFilesystems Recovery Method: sequential
Shared Tape Resources Filesystems Recovery Method: sequential
Application Servers: app2Filesystems Recovery Method: sequential
Highly Available Communication LinksFilesystems Recovery Method: sequential
Miscellaneous DataFilesystems Recovery Method: sequential
Automatically Import Volume Groups: falseFilesystems Recovery Method: sequential
Inactive Takeover: falseFilesystems Recovery Method: sequential
Cascading Without Fallback: falseFilesystems Recovery Method: sequential
9333 Disk Fencing: falseFilesystems Recovery Method: sequential
SSA Disk Fencing: falseFilesystems Recovery Method: sequential
Filesystems mounted before IP configured: false

Run time parameters:

Node Name: **HAdb2**
Debug Level: high
Host uses NIS or Name Server: false
Format for hacmp.out: Standard

Node Name: **HAdb1**
Debug Level: high
Host uses NIS or Name Server: false
Format for hacmp.out: Standard

Then synchronize and verify the clusters. The setup is complete.

Conclusion

High availability can be achieved with DB2 UDB with excellent results. We succeeded in achieving the goal of this POC because the measured failover time was approximately 60 seconds in most cases. Starting the database itself took only one second. Most of the time was consumed in moving resources between servers and especially the disks. In the Version 5.1 of HACMP, this time is eliminated with the use of the enhanced concurrent access mode for the disks. Overall, the mutual takeover scenario is also a very good solution for the case where the environment needs two separate databases.

Acknowledgements

I would like to thank Alan Barnes for his assistance and advice in the writing of this article.

I would like to thank Enzo Cialini, manager of the HA system test team in the Toronto lab, for his continuous and very valuable help during this effort.

I also thank Glenn Miller for his outstanding support in all matters concerning AIX and HACMP setup.