

Platform: DB2 UDB for Linux, UNIX, Windows

Disaster Recovery and DB2/AIX: A User Experience

Bill Gallagher

Senior Database Administrator / Phoenix Life Insurance

Session: D10

Wednesday, May 25, 2005 12:30pm - 1:40pm



This presentation will describe the Disaster Recovery solution developed and implemented for Phoenix Life Insurance's DB2 for AIX environment. It will detail the logical AIX environment at our production and DR sites, the specifics of our DR strategy for DB2 on AIX, and the reasons why we implemented this particular solution. It will also describe the DB2 commands and other utilities that are an important part of the solution.

Bill Gallagher is a senior Database Administrator for Phoenix Life Insurance Company in Hartford, CT. Bill has over 22 years experience in the Information Technology field, the last 16 as a Database Administrator supporting IDMS, MS SQL Server and DB2 UDB for both the OS/390 and AIX platforms. Bill's primary responsibilities include application DBA support for the company's DB2 databases on both the mainframe and mid-tier platforms. Recent projects include the development and implementation of the DB2/AIX solution for Disaster Recovery, and overseeing the migration of the DB2/AIX platform from Phoenix's in-house data center to an external vendor's data center due to IT Infrastructure outsourcing.

Bill Gallagher, DBA
Phoenix Life Insurance
One American Row
Hartford, CT 06102
(860) 403-6327
bill.gallagher@phoenixwm.com

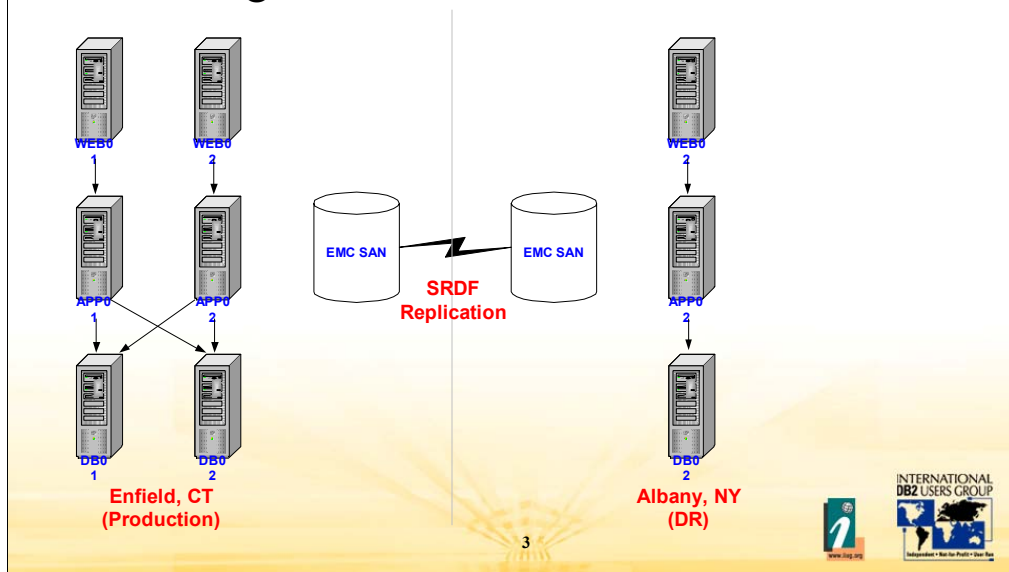
Disaster Recovery and DB2/AIX: A User Experience

- PNX's Logical AIX Environment (Production and DR)
- PNX's Disaster Recovery Solution for DB2/AIX
- DB2 Commands and Utilities Used in the Solution
- Challenges Faced and Successes
- What's Next? and References



- I PNX's Physical DB2 Environment (Production and DR)
- II PNX's Disaster Recovery Solution for DB2/AIX
 - Pre-DR Environment and Premises
 - Common DR Approaches for DR
 - PNX's DR Approach
 - Examples of Required Filesystems and Directories
- III DB2 Commands and Utilities Used in the Solution
 - Background on DB2 Crash Recovery
 - Two Important LSN Values You Need to Know
 - Commands and Utilities Used
 - Flow of Database Backup/Cleanup Script
 - What to Do at Disaster Recovery Time
- IV Challenges Faced and Successes
 - How to Move Existing Databases into New Architecture
 - Our Timeline of Events
- V Lessons Learned and References
 - What's Happened Since We Implemented This Solution?
 - New Features in DB2 v8
 - Reference

PNX's Logical AIX Environment



This is a very high-level overview of our logical AIX architecture at both our Production data center in Enfield, CT, and our Disaster Recovery data center located near Albany, NY. The two data centers are located approximately 90 miles apart (driving distance), or 59 nautical miles.

I've intentionally left out details about the physical characteristics of the hardware involved for both the AIX servers themselves and the EMC SAN (storage area network) because they are not relevant within the context of this presentation.

All AIX servers were originally running AIX 4.3.3 in the production data center in Enfield, and AIX 5L in the DR data center in Albany. The production servers have since been upgraded to AIX 5L as well.

The two production web servers are basically clones of each other, containing web content. A load balancing switch (not shown) sits in front of the two web servers, and directs traffic to either WEB01 or WEB02.

The two production application servers are also basically clones of each other, running WebSphere 5.0. Another load balancing switch (also not shown) sits in front of the two application servers directing traffic. However, this switch is used primarily to handle a fail-over situation in case one of the application servers is down. In general APP01 primarily communicates with WEB01, and APP02 primarily communicates with WEB02.

The two production database servers will be discussed in a little more depth in the next slide.

PNX's Logical AIX Environment

- Two production database servers in Enfield, CT
 - Separate and distinct, each running DB2 v7.2 EE fixpak 7
 - No clustering or fail-over between the two servers
 - Data Warehouse databases reside on DB01
 - OLTP databases reside on DB02
 - No affinity to application servers (i.e. traffic could come from either APP01 or APP02)

While the two web servers and the two application servers are essentially clones of each other, the two database servers are separate and distinct servers with no redundancy, clustering, or fail-over capabilities.

The two database servers are currently running DB2 v7.2 Enterprise Edition at fixpak 7.

Originally, the two database servers were intended to spread the workload evenly across the two servers based on the anticipated processing against the databases hosted on each server. There was a fairly even mix of databases across the two servers, both large and small databases supporting mostly OLTP applications. Workload was coming from the two application servers fairly evenly to each database server. However, it became apparent over the course of time that our anticipated workload estimates for the two database servers were much higher than we were actually experiencing, so we had excess capacity available on both servers, which in turn allowed us some flexibility in making changes to our database landscape.

Soon after our Data Warehousing application went into production in late 2003, we initiated an “Isolated Environment” project in which we dedicated the DB01 server exclusively to the Data Warehouse, and moved all other databases to the DB02 server. This migration was completed in August 2004.

As a result of this migration, the Data Warehouse databases are now hosted entirely on the DB01 server, and all other DB2/AIX databases are hosted on the DB02 server. Because the Data Warehouse application does not have a WebSphere front-end, all database traffic coming from the two application servers are currently going exclusively to the DB02 database server.

PNX's Logical AIX Environment

- One physical Disaster Recovery server in Albany, NY
 - Logically “carved” into four LPAR's (logical partitions)
 - WEB02 is cloned from Enfield
 - APP02 is cloned from Enfield
 - DB02 is cloned from Enfield
 - Fourth LPAR is for another unrelated application
- DR Server is normally physically powered down
- EMC SAN is always up and replicating between Enfield and Albany

5



In our Albany, NY Disaster Recovery data center, we have a number of “Wintel” servers and one RS/6000 AIX server that are dedicated solely to DR.

The RS/6000 server is logically carved into four separate logical partitions, or LPAR's. Three of these LPAR's are the DR counterparts of the WEB02, APP02, and DB02 production servers in Enfield. The fourth partition is used for DR recovery of another production AIX server in Enfield that is unrelated to our production web environment.

It's important to note here that the DR instances of WEB02 and APP02 in Albany are essentially clones of WEB02 and APP02 in Enfield, which in turn are clones of WEB01 and APP01 in Enfield. However, the DR instance of DB02 in Albany is a clone of the DB02 production server in Enfield, which is separate and distinct from the other production database server, DB01. There is no specific clone of DB01 at the DR site in Albany.

We have accommodated for this by planning to recover all the databases in scope of DR, whether they reside on the DB01 or DB02 production databases servers in Enfield, to only the DB02 DR database LPAR in Albany.

Another item of interest: the DR RS/6000 server in Albany is normally physically powered down. Because it is dedicated to DR and serves no other purpose, there is no need to keep it physically powered up, consuming power and causing normal “wear and tear” on the physical internal components of the server. The normal powering up of the box in the event it is needed is on the order of 10-15 minutes, which is fully acceptable for our purposes.

However, the EMC SAN is always up and replicating via SRDF (Symmetrix Remote Data Facility) between Enfield and Albany, so the required DASD components in Albany are readily available and synchronized with our production site when they are actually needed.

PNX's Disaster Recovery Solution for DB2/AIX

- Pre-DR environment: all database backups and archive logs sent directly to TSM using the DB2 TSM api exit
- We are operating under the premise that any DB2/AIX databases that need to be recovered in < 72 hours cannot have their “primary” backups stored in TSM
 - We cannot guarantee that the TSM server (AIX) will be fully recovered and functional within the first 72 hours of disaster
- We needed to rethink our backup and archive log strategy for all DB2/AIX databases in scope of DR

Prior to constructing our DR environment, which went “live” in March 2004, all of our DB2 database backups and archive logs went directly to a TSM (Tivoli Storage Manager) server, using the DB2 TSM api. This was done using the “USE TSM” keywords in the backup command for database backups, and by using the “db2uext2” user exit for archive logs.

However, once we started planning for DR, we quickly realized that within our environment it might not be reasonable to expect that our TSM server would be fully recovered and functional within the first 72 hours following a disaster. Seventy-two hours was our SLA (Service Level Agreement) with the business areas for getting a number of our critical systems up and running in the event of disaster. Even if the TSM server was fully recovered, it could be expected to be burdened with a lot of recovery activity, which in turn would have an extremely negative impact on TSM response time.

This meant that any applications and/or databases that needed to be recovered and be fully operational within 72 hours of disaster could not be dependent on TSM for any aspect of its recovery. For DB2, this meant that we needed to rethink our database backup and archive log strategy in order to take TSM out of the equation.

PNX's Disaster Recovery Solution for DB2/AIX

- Common DR Approaches for DB2 - Log Shipping
 - A copy of the production database is “waiting” at the DR site in rollforward pending state
 - As logs are archived from the production database, they are copied to the DR site and applied to the standby database via rollforward utility
 - This can be set up with minimal impact to the production system
 - Only archive logs are sent to DR site; committed transactions in active logs may never be sent to DR site in the event of disaster
 - Does not account for non-logged activity, such as “not logged initially” processing or NONRECOVERABLE loads

A “remote standby database” is a database that is residing in a remote location that is typically in an offline state, ready to become operational in the event of a failure at the production site.

In the case of a remote standby database using a log shipping approach, the database would have been initially created by restoring from a database backup, or using a split-mirror technique that would have copied all required database files to disks at the remote location. The remote database would be “waiting” in a rollforward pending state.

In this approach, the archive log user exit would be enhanced to copy the archive logs to the remote site as they are created. Each time a new archive log gets shipped to the remote site, a “ROLLFORWARD TO END OF LOGS” command can be used to apply the most recent set of transactions to the standby database. At the time of disaster recovery, a “ROLLFORWARD TO END OF LOGS AND COMPLETE” command would be used to complete the recovery, remove the rollforward pending state, and make the database available for use.

While this approach allows for the least amount of impact to the production system, it does allow for a number of exposures for data loss. First, since only archive logs are being sent to the remote site, any transactions that reside in the active logs at the time of disaster will never be sent to the remote site, and will in essence be lost. Second, only logged activity is being shipped to the remote site via the archive logs; any non-logged activity is not accounted for in the standby database. Any applications that use “NOT LOGGED INITIALLY” or load utilities with the “NONRECOVERABLE” option will cause those tables to be marked as inaccessible when encountered in the logs. Using the load utility with the COPY NO option will cause the tablespaces to be placed in a restore pending state. Using the load utility with the COPY YES option will require that the load copy file be shipped to the remote site within the same filesystem/directory/file name structure in order for the tables to be accessible following the rollforward processing.

PNX's Disaster Recovery Solution for DB2/AIX

- Common DR Approaches for DB2 - Mirrored Disks
 - All disk components that make up a database (database directory, tablespace containers, active and archive logs, etc.) are synchronously replicated to a standby database at the DR site
 - All that needs to be done at DR site is to start the DB2 instance and database, which initiates crash recovery to bring the database to a usable and consistent state
 - This can provide the most complete recovery of a database at the DR site in the shortest time possible
 - Can have severe performance impact since all disk writes are synchronously replicated between production and DR sites

In the case of a remote standby database using mirrored disks, all disk components that make up a database would be synchronously mirrored, or replicated, to disks at the remote location. This would include the database directory, tablespace containers, active logs, archive logs, and external routines such as stored procedures and user defined functions (UDF's).

The DB2 instance at the remote site would be stopped, so there would be no application access to the data on the remote disks until the disaster recovery is initiated. At that point, the instance at the remote site would be started and crash recovery would be initiated to bring the database back to a consistent state with all uncommitted transactions as of the time of disaster rolled back.

Since the remote site would have all of the completed writes synchronously replicated from the production site, all of the logs (active and archive) and the tablespace containers will be completely up-to-date. This allows for the potential to provide the most complete recovery of a database in the shortest time possible.

This comes with a potentially steep price, though. Since all disk writes for the active logs, archive logs, and tablespace containers would be delayed at the production site until the mirrored disks are updated at the remote site, there would be a severe performance impact on all update transactions, particularly for OLTP databases which by nature are update-intensive.

PNX's Disaster Recovery Solution for DB2/AIX

- Common DR Approaches for DB2 - Mirrored Logs
 - A hybrid of the Log Shipping and Mirrored Disk approaches in that archive logs are copied to the DR site as they are archived from the production database, and the active logs are synchronously replicated between the production and DR sites
 - Gets around the “lost transaction” problem inherent with the log shipping approach as all active and archive logs are available at the DR site
 - As with the Log Shipping approach, does not account for any non-logged processing

An alternative to the log shipping and mirrored disks approach would be a remote standby database using mirrored logs. This is actually a combination of the log shipping and mirrored disk approaches in which the remote database would be waiting in a rollforward pending state. However, in addition to the archive log user exit copying logs from the production site to the remote site as they are archived, the active logs are synchronously replicated between the two sites.

As with the log shipping approach, a periodic “ROLLFORWARD TO END OF LOGS” command would be issued at the remote site to apply the most recent set of logged transactions to the remote database. At the time of disaster recovery, a “ROLLFORWARD TO END OF LOGS AND COMPLETE” would be used to complete the recovery, remove the rollforward pending state, and make the database available for use.

This gets around the “lost transaction” problem inherent with the log shipping approach in that all log files should be available at the remote site at the time of disaster.

However, it still has many of the same exposures for data loss as the log shipping approach in that any non-logged activity will not be reflected at the remote site. Also, there will be a performance impact at the production site as all writes to the active log files will be delayed until the mirrored active logs are updated at the remote site. For high-volume OLTP databases, this can have a noticeable impact upon performance.

PNX's Disaster Recovery Solution for DB2/AIX

- PNX's DR Approach is a variation of the Mirrored Logs approach
 - All database files are replicated via hardware between the production and DR sites, some synchronously and others via snapshots
 - Addresses all the exposures of the Log Shipping and Mirrored Logs approaches
- All physical database components required for database recovery must reside on disk and be replicated to the DR site
 - Database files (i.e. tablespaces)
 - Backups
 - Active and archive logs

The approach that we adopted is a variation of the mirrored logs approach in that all database files are replicated via hardware from the production site to the remote DR site. We use a mix of synchronous mirroring/replication and snapshot, or time-based replication in order to accomplish this. Our solution addresses all the exposures of the log shipping and mirrored logs approaches.

As mentioned earlier, we needed to remove any dependencies that we had on TSM for recovering our databases at the remote site in the event of disaster, so we had to come up with a strategy of ensuring that all physical database components required for database recovery would be immediately available on disk at the time of DR. This included the following database components:

- Database files, i.e. tablespaces (data, index, LOB, temp, catalog, etc.)
- Backups
- Active Logs
- Archive Logs
- Instance filesystems

PNX's Disaster Recovery Solution for DB2/AIX

- Different physical database components have different replication requirements
 - Database files (i.e. tablespaces) - weekly replication/snapshot
 - Just a “placeholder” for space
 - Backups - daily replication/snapshot (6:00am)
 - Active and archive logs - synchronous replication
- Database recovery strategy assumes that each database in scope of recovery will be restored to the point in time as of disaster
 - Database restored to most current full backup on disk
 - Rollforward to end of logs (and complete)

11



In our DR strategy for DB2/AIX, different physical database components have different replication requirements. These requirements range from weekly snapshots to daily snapshots to synchronous mirroring of selected AIX filesystems which have been specifically created for the DB2 instances and databases that have been defined by the business as being in scope of DR. By “in-scope of DR”, we are referring to those databases that need to be guaranteed to be recovered and fully operational within the first 72 hours of disaster. In reality, our DR strategy for DB2/AIX can fully recover DB2 databases within minutes or hours of a disaster.

The three different groups of database components, and their replication requirements, are:

- Database files (i.e. tablespaces and DB2 instances) - weekly snapshot replication - we do not need the database files themselves to be up to date, we just need to have the space allocated as a “placeholder”. The DB2 instance software would also be replicated weekly, which should not be an issue as we would not expect it to change that often. The only exposure would be that the “db2diag.log” file would not be current, but we did not see that as a potential problem.
- Database backups - daily snapshot replication at 6:00am - we have a job scheduled in the DB2 journal which will take full online backups of the database to disk at approximately 5:00am, which gives us sufficient time to have all the backups completed by 6:00am.
- Active and archive logs - synchronous replication - this ensures that all active and archive logs are current at the remote DR site at all times

Our DR strategy for DB2/AIX assumes that each database in scope of recovery at the DR site will be restored to the point-in-time as of disaster. This will be accomplished by recovering the database to the most current full backup on disk (taken at 5:00am that morning), and rollforwarding to the end of the logs (and complete).

PNX's Disaster Recovery Solution for DB2/AIX

- AIX filesystems required to support this are:
 - /db2tablespacen
 - /db2indexspacen
 - /db2tempspacen
 - /db2backupsn
 - /db2activelogsⁿ
 - /db2archivelogsⁿ
- Need different filesystem names for databases residing on DB01 and DB02 because they will all be mounted to a single DR database server, so names must be unique

12



In order to support this strategy, we defined three separate AIX volume groups, each with their own replication requirements. Within each of the volume groups, we defined the following filesystems:

Volume group “db2vga” (weekly snapshot replication)

- /db2tablespacen
- /db2indexspacen
- /db2tempspacen
- /(instance filesystems)

Volume group “db2vgb” (daily snapshot replication)

- /db2backupsn

Volume group “db2vgc” (synchronous replication)

- /db2activelogsⁿ
- /db2archivelogsⁿ

Note the suffix of “n” at the end of each of the db2 filesystem names. This “n” is actually replaced with either a 1 or a 2, depending on which of the two databases servers (DB01 or DB02) the filesystems are defined on. The reason for this is that we have two different database servers in production, but only one at the remote DR site. The plan is for all databases in scope of DR on both production servers to be recovered to a single server at the DR site. Because we cannot have multiple filesystems with the same name on a single system, we must make each set of filesystem names unique across the two production servers.

PNX's Disaster Recovery Solution for DB2/AIX

- Examples of filesystems and directories required for database MYDB2DB
 - /db2tablespace2/MYDB2DB/TSD_MYDB2DB_01
 - /db2indexspace2/MYDB2DB/TSI_MYDB2DB_01
 - /db2tempespace2/MYDB2DB/SQLT0001.0
 - /db2backups2/MYDB2DB/current/MYDB2DB.0.db2inst1.NODE0000.CATN0000.20041221051006.001
 - /db2backups2/MYDB2DB/current-1/MYDB2DB.0.db2inst1.NODE0000.CATN0000.20041220051005.001
 - /db2activelogs2/MYDB2DB/S0002164.LOG
 - /db2archivelogs2/MYDB2DB/NODE0000/S0002153.LOG



Each of the six filesystems have a set of mandatory directories that also need to be created to support this solution. They are:

/db2tablespacn
/dbname

/db2indexspacn
/dbname

/db2tempspacn
/dbname

/db2backupsn
/dbname/current - where the most current database backup is stored
/dbname/current-1 - where the previous database backup is stored

/db2activelogs
/dbname

/db2archivelogs
/dbname/NODE0000 - this is the “default” directory structure required by the “db2uext2” exit for archiving DB2 logs to disk

DB2 Commands and Utilities Used in the Solution

- First, some background on DB2 Crash Recovery
 - Crash recovery is the process in which a database is moved back into a consistent and usable state following an unexpected failure
 - Committed transactions that have not yet been externalized to disk are completed (i.e. the “redo” process)
 - Incomplete, or uncommitted transactions are rolled back (i.e. the “undo” process)
 - The Log Control File (sqlogctl.lfh) is used to determine the starting point for crash recovery in the log files

Before we get into the DB2 commands and utilities used in our DR solution, it's good to have a basic understanding of what DB2 crash recovery is and how it works.

Crash recovery is the automatic recovery of a database if a failure occurs while there are active units of work at the time of failure. Crash recovery will rollback any incomplete, or uncommitted, transactions and will also ensure completion of any committed transactions that were not yet externalized to disk before the failure occurred. These two processes are known as the “undo” and “redo” processes. After successful completion of crash recovery, the database will be available in a consistent and usable state

Crash recovery can either be an automated or a manual process based on the value of the AUTORESTART database configuration parameter. AUTORESTART(ON) will automatically perform crash recovery upon the first connection to the database after failure. AUTORESTART(OFF) will require the issuing of a RESTART DATABASE command after the failure in order to initiate crash recovery.

The “undo” process will rollback all incomplete, or uncommitted transactions at the time of failure by processing backwards through the active log files until all changes performed by those transactions have been completely backed out.

The “redo” process will reprocess all changes in the active log files that have not been externalized to the physical tablespace containers. The changes will be applied to pages in the DB2 buffer pools, which will then be written back to the tablespace containers asynchronously via the database's page cleaner processes.

The starting and ending points for log processing for the crash recovery process are stored in the database's Log Control File (sqlogctl.lfh).

DB2 Commands and Utilities Used in the Solution

- There are two important values that are critical to DB2 crash recovery and roll-forward processing
- MIN_BUFF_LSN
 - A pointer to the log record that contains the oldest changed database page in the bufferpool which has not yet been externalized to disk
 - Represents the oldest log record that would need to be redone for crash recovery - the start of the “redo” process
- LOW_TRAN_LSN
 - A pointer to the log record that is associated with the oldest uncommitted transaction for the database
 - Represents the oldest log record that will need to be undone for crash recovery - the end of the “undo” process

There are two important values that control the starting and ending points for DB2 crash recovery. They are MIN_BUFF_LSN and LOW_TRAN_LSN.

MIN_BUFF_LSN represents a pointer to the log record that contains the oldest changed database page in the bufferpool that has not yet been externalized to disk. It's important to understand that committed transactions may not have been written to the physical tablespace containers on disk at the time of failure, i.e. the changes have only been applied to the bufferpool pages in memory. This is why DB2 needs to keep track of the oldest log record that references a dirty bufferpool page: to ensure that these changes which resided only in memory can eventually be written to disk during the crash recovery “redo” process.

The value of MIN_BUFF_LSN is indirectly dependent on the SOFTMAX database configuration parameter, which determines how often page cleaners are triggered to write dirty bufferpool pages to disk. A discussion of SOFTMAX and the implications of tuning it is beyond the scope of this presentation.

LOW_TRAN_LSN represents a pointer to the log record that is associated with the oldest uncommitted transaction for the database. Unlike MIN_BUFF_LSN, it's important to understand that dirty bufferpool pages from uncommitted transactions may be written to the physical tablespace containers. DB2 needs to keep track of these uncommitted transactions to ensure that all these changes can be backed out during the crash recovery “undo” process.

DB2 Commands and Utilities Used in the Solution

- The minimum of MIN_BUFF_LSN and LOW_TRAN_LSN represents the oldest log record that will be needed to participate in crash recovery
- The two values are maintained in the Log Control File for the database
- The Log Control File is included in each DB2 backup image in order to associate the backup with the specific log files which are relevant to that backup image, and to establish the point in the logs where roll forward processing begins following a restore of a backup



In short, MIN_BUFF_LSN points to the log record that will be used as the start of the “redo” process during crash recovery, and LOW_TRAN_LSN points to the log record that will be used as the end of the “undo” process during crash recovery.

These two values are saved in the Log Control File (sqllogctl.lfh) every time a log file is filled, or at “soft checkpoint” time, i.e. whenever the SOFTMAX threshold is hit. It should be noted that the values of MIN_BUFF_LSN and LOW_TRAN_LSN in the Log Control File may differ from the the most current values in database global memory, which are constantly updated.

A copy of the Log Control File is included at the end of each DB2 database backup image in order to associate that backup image with the specific log files which are relevant to that backup. It also establishes the starting point in the logs where roll forward processing begins following the restore of a backup image.

DB2 Commands and Utilities Used in the Solution

- Since backups and archive logs are now residing on disk, we need a way to clean them up, keeping only the most current files required for recovery residing on disk
 - We decided to keep the most recent two full sets of backups on disk
- Backups are done via an AIX shell script that performs cleanup of the backup and archive log directories as part of the backup process
 - A single script is used for all databases, with database name and filesystem suffix (1 or 2) passed as parameters

Since database backups and archive logs were now residing on disk as opposed to being shipped directly to TSM via the DB2/TSM api, we needed to come up with some sort of automated mechanism to clean up older backups and logs that were no longer needed on disk. We only wanted to keep the most current backup images and archive logs on disk for quick recovery in the event of a disaster. Just to be on the safe side, we decided to keep the two most recent full sets of backups and archive logs on disk. Anything older than that should be removed from disk in order to prevent us from consuming all our of disk space. However, we did not want to delete any backup images or archive log files from disk unless we had a way of ensuring that they were also backed up to TSM.

We have a set of TSM incremental backup jobs that run every night that “sweep up” all files that have been changed during the day. By checking TSM, we should be able to determine if a particular disk file has been backed up, and if it is “safe” to delete it from disk.

We ended up writing an AIX shell script that performs cleanup of the database backup and archive log directories as part of the backup process. It invokes a handful of TSM command line commands as well as DB2 command line utilities to help determine which files are still needed, which files are candidates for deletion from disk, and which of these candidate files have been backed up to TSM via the nightly “sweep” jobs.

Because we have well defined set of filesystem and directory naming standards, the backup/cleanup script was written such that it needs to be passed only two parameters to be able to process any database that is in scope of DR. There two parameters are the filesystem suffix (“1” for server DB01, or “2” for server DB02), and the database name.

DB2 Commands and Utilities Used in the Solution

- Commands and utilities used by the backup/cleanup script
 - DSMC QUERY BACKUP - TSM command
 - Displays a list of backup versions of a specified file
 - “dsmc query backup -inactive /db2archive2/MYDB2DB/NODE0000/S0002118.LOG”
 - DB2CKBKP - DB2 command line utility
 - Tests the integrity of a backup image, and can also display metadata stored in the backup header in order to determine information about a backup image
 - “db2ckbcp -l /db2backups2/MYDB2DB/current-1/MYDB2DB.0.db2inst1.NODE0000.CATN0000.20041220051005.001 | grep LSN”

There are a handful of TSM and DB2 commands and utilities that were very useful in our Disaster Recovery solution for DB2.

The first was the “dsmc query backup -inactive” TSM command line command, which will pass the fully qualified (i.e. full directory and filename specification) name of a disk file and return information about the backup versions of that file that are stored at the TSM server. This command came in quite handy for helping us “clean up”, or delete older database backup images from disk while ensuring that we still had a copy of that image in TSM should we need to restore the database to an older backup.

The second was the “db2ckbcp” DB2 command line utility. This utility will test the integrity of a backup image and determine whether it can be restored. It can also be used to display metadata stored in the backup header in order to determine specific information about a particular backup image. Such metadata includes DMS tablespace header information, media header information, object header information, and log file header data. We used the latter option in order to extract the MIN_BUFF_LSN and LOW_TRAN_LSN values from the Log Control File stored in the backup image.

DB2 Commands and Utilities Used in the Solution

- Commands and utilities used by the backup/cleanup script
 - DB2FLSN - DB2 command line utility
 - Returns the name of the file that contains the log record identified by a specified LSN
 - Command must be issued from the same directory that the Log Control File (sqllogctl.lfh) resides in
 - You can create a link to the file from another directory - in our case, we link to this file from /db2backupsn/dbname/current-1
 - “db2flsn 0009EE7DC1B6”

The third DB2 command line utility that we used in our solution was the “db2flsn” (Find Log Sequence Number) utility. Passing a 12-digit hex LSN to this utility will result in the utility using information in the Log Control File (sqllogctl.lfh) to return the name of the log file that contains the log record associated with the given LSN.

It’s important to note that the “db2flsn” utility uses the LOGFILSIZ database configuration parameter in determining it’s result. DB2 keeps track of the three most recent values of LOGFILSIZ, as well as the name of the first log file associated with each LOGFILSIZ value. This allows the utility to work correctly when the LOGFILSIZ parameter changes.

We used the “db2flsn” utility to find the name of the oldest log file needed to participate in roll-forward recovery for a backup image by determining the minimum of MIN_BUFF_LSN and LOW_TRAN_LSN, and passing that LSN value to the utility. Once we knew the name of this log file, all older archive log files were identified as candidates for deletion during the disk cleanup process.

One other note: the “db2flsn” utility must be executed from the same directory in which the Log Control File resides. We worked around this by creating an AIX link to the Log Control File for the database from the directory where we intended to execute this utility, which in our case was the /db2backupsn/dbname/current-1 directory.

DB2 Commands and Utilities Used in the Solution

- Flow of backup/cleanup script:
 - Move the backup image in the /db2backupsn/dbname/current directory to /db2backupsn/dbname/current-1 directory
 - “dsmc query backup -inactive” command will check the backup images in the ../current-1 directory to see if they have been backed up via TSM
 - If they have, then delete them from disk
 - If they have not, then retain them on disk

20



Example of backup/cleanup script:

Backup and archive log directories (before executing script):

```
/db2backups2/MYDB2DB/current-1/  
MYDB2DB.0.db2inst1.NODE0000.CATN0000.20041225051320.001
```

```
/db2backups2/MYDB2DB/current/  
MYDB2DB.0.db2inst1.NODE0000.CATN0000.20041226051319.001
```

```
/db2archivelogs2/MYDB2DB/NODE0000/S0002122.LOG  
/db2archivelogs2/MYDB2DB/NODE0000/S0002123.LOG  
/db2archivelogs2/MYDB2DB/NODE0000/S0002124.LOG
```

Backup/cleanup script executed at 5:10am on 12-27-2004

The backup image in the ../current directory is moved (not copied) to the ../current-1 directory

```
/db2backups2/MYDB2DB/current-1/  
MYDB2DB.0.db2inst1.NODE0000.CATN0000.20041225051320.001
```

```
/db2backups2/MYDB2DB/current-1/  
MYDB2DB.0.db2inst1.NODE0000.CATN0000.20041226051319.001
```

```
/db2backups2/MYDB2DB/current/  
empty
```

DB2 Commands and Utilities Used in the Solution

- Flow of backup/cleanup script (continued):
 - Perform a full online backup of the database to the ../current directory
 - Determine the name of the oldest archive log file that is associated with the oldest backup image residing on disk
 - “db2ckbcp” utility will identify the “Low Tran LSN” and “Min Buff LSN” values
 - The minimum of these two values will identify the earliest log record required for recovery

Example of backup/cleanup script (continued):

Use the “dsmc query backup -inactive” command to check if the backup images in the ../current-1 directory have been backed up to TSM

The ../20041225051320.001 backup image was backed up to TSM at 10:47pm on 12-26-2004, so delete it from ../current-1

The ../20041226051319.001 backup image has not yet been backed up to TSM (we just moved it there moments ago, so keep it)

Take a full online backup of MYDB2DB to the ../current directory

Run the “db2ckbcp” utility against the oldest backup image in ../current-1

```
db2ckbcp -l /db2backups2/MYDB2DB/current-1/..20041226051319.001 | grep LSN
```

Returns (among other LSN values):

```
LOW TRAN LSN           = 0009 F4D4 86E1
MIN BUFF LSN           = 0009 F4D4 0043
```

Run the “db2flsn” utility with the minimum of these two LSN’s
db2flsn 0009F4D40043

Returns:

Given LSN is contained in log file S0002123.LOG

DB2 Commands and Utilities Used in the Solution

- Flow of backup/cleanup script (continued):
 - “db2flsn” utility will identify the name of the archive log file that contains this earliest log record required for recovery
 - “dsmc query backup -inactive” command will check the archive log files on disk to see if they have been archived via TSM
 - If they have and they are older than the oldest log file we need for recovery, then delete them from disk

Example of backup/cleanup script (continued):

Use the “dsmc query backup -inactive” command to check if the archive logs for the database have been backed up to TSM

/db2archive/logs2/MYDB2DB/NODE0000/S0002122.LOG was backed up to TSM at 11:19pm on 12-25-2004; it's older than S0002123.LOG, so delete it

/db2archive/logs2/MYDB2DB/NODE0000/S0002123.LOG and S0002124.LOG were backed up to TSM at 11:06pm on 12-26-2004; but we still need them, so keep them

/db2archive/logs2/MYDB2DB/NODE0000/S0002125.LOG has not yet been backed up to TSM; it was created when we did the full online backup moments ago, so keep it

Backup and archive log directories (after executing script):

/db2backups2/MYDB2DB/current-1/
MYDB2DB.0.db2inst1.NODE0000.CATN0000.20041226051319.001

/db2backups2/MYDB2DB/current/
MYDB2DB.0.db2inst1.NODE0000.CATN0000.20041227051250.001

/db2archive/logs2/MYDB2DB/NODE0000/S0002123.LOG
/db2archive/logs2/MYDB2DB/NODE0000/S0002124.LOG
/db2archive/logs2/MYDB2DB/NODE0000/S0002125.LOG

DB2 Commands and Utilities Used in the Solution

- What to do at Disaster Recovery time?
 - All components for recovering the database should be present at the DR site on disk in the six filesystems for DB2 databases and logs
 - After starting the database server and DB2 instances, each database to be recovered needs to be recovered to the most recent backup and rollforwarded to the end of logs
 - db2 restore database **dbname** from /db2backupsn/**dbname**/current
 - db2 rollforward database **dbname** to end of logs and complete
 - At this point, the database will be restored to the point of failure, and available for processing

At Disaster Recovery time, everything should be in place to ensure a complete and easy recovery of all databases in scope of DR. All necessary filesystems, backup images, and log files needed for database recovery should be available at the DR site to recover the database up to the point of failure.

After the AIX administrators start the database server and mount all the filesystems to the server, the DB2 instances should be able to start successfully. Once the instances are started, all that is needed is to issue the following two commands for each database in scope of DR:

```
db2 restore database dbname from /db2backupsn/dbname/current  
db2 rollforward database dbname to end of logs and complete
```

This will restore each database to the most current backup stored on disk, and rollforward the transaction logs to the point of failure. Any uncommitted transactions as of the point of failure will be rolled back, and the database will be left in a consistent and usable state.

Challenges Faced and Successes

- How to move existing databases into new disk architecture?
 - Need to determine how much space to allocate for each of the new filesystems
 - /db2tablespac n - derived from space already allocated for table tablespaces
 - /db2indexspac n - derived from space already allocated for index tablespaces
 - /db2temp $spac$ n - as large as you want or need it (usually near-empty)
 - /db2backups n - as large as 3x the size of a full backup (to hold previous two full backups plus contingency in case nightly TSM backup does not run)
 - /db2activelogs n - $(\text{LOGPRIMARY} + \text{LOGSECOND}) * \text{LOGFILSIZ} * 4K$

Once we had our strategy all planned out, we were then faced with the task of implementing it. Implementation was a fairly straightforward process in that all it really involved was moving the databases in scope of DR into the new disk and filesystem architecture. But first, we had to determine how to size the new filesystems so that they were properly allocated with an appropriate amount of space.

Some of the filesystems were fairly easy. For instance, sizing the tablespace and indexspace filesystems was pretty straightforward in that we just needed to look at how much space was already allocated for tablespaces and indexspaces for those databases, and then add an additional percentage of free space to allow for future growth.

The temp $spac$ tablespace filesystem was also fairly easy to size in that we just needed to allocate a large enough “chunk” of space to handle what we estimated to be our largest need for temp $spac$ at any single point in time. Typically, this filesystem would be near empty as most of our transactions would use only a small amount of temp $spac$ at any one point in time.

The backups filesystem was a little trickier in that we needed to determine how big the backup images were for each of our databases in scope, and then multiply that by a factor of 3x. We chose 3x because we knew we were keeping the two most recent database backup images on disk, but also had to allow for some contingency in case the nightly TSM incremental backup sweeps failed to run, thus preventing us from deleting an older set of database backups from disk.

The active logs filesystem was very easy to size in that we could determine the maximum amount of active log files and the size of each log file from the appropriate database configuration parameters for each database. Again, we would add an additional percentage of free space to allow for future growth.

Challenges Faced and Successes

- /db2archive logs_n - need to be sized to hold the maximum number of archive logs that can be generated over a two day period
 - Wrote an AIX shell script to extract this data from TSM
 - dsmc query archive “/dbname/*” -subdir=yes | grep API
 - Imported the extracted data into an Excel spreadsheet for summarization and analysis
 - Be aware of special monthly, quarterly, or yearly processing which may generate an extraordinarily higher number of archive logs than normal daily processing
 - Analyzing this information also gave us some good insight into how much synchronous replication we would be doing between our production data center and our DR data center for both active and archive logs

Sizing the archive log filesystem was by far the most difficult step of this part of the implementation. Since the number of archive logs created is directly dependent on the transaction workload for the database, we needed to get a handle on how much log activity was being generated for each database on a daily basis. Since our archive logs were being shipped directly to TSM via the userexit, we needed to find a way to get information out of TSM regarding the number of log files being archived, as well as the size of each log file. It's important to note that not all log files that get archived are full; DB2 will close and truncate any unused space in the active log file whenever the database is deactivated, when an online backup is run against the database, or when the ARCHIVE LOG command is issued against the database.

In talking to our TSM administrator, we were able to determine that the “dsmc query archive” command would help us extract the information we needed from TSM concerning our archive logs. Executing the “dsmc query archive” command as documented above produced a fixed format text file which we were able to ftp to a Windows workstation and import into an Excel spreadsheet. We were then able to summarize and analyze our daily archive log activity for each database, thus allowing us to appropriately size the archive log filesystem.

We also derived a secondary benefit from this exercise in that by understanding how much archive log activity we were generating per database per day, we also knew how much synchronous replication would be going on between our production data center and our DR data center for both the archive log and active log filesystems. Since both active logs and archive logs were to be replicated synchronously between the two data centers, we were able to take the numbers for archive log space consumed per day, multiplied by 2 (because each archive log was once an active log), to determine how much synchronous replication we could expect. Because the data from the “dsmc query archive” command was timestamped, we also knew how big our largest bursts of synchronous activity might typically be.

Challenges Faced and Successes

- Create the new AIX volume groups and filesystems
 - Three separate volume groups because of three different sets of replication requirements
 - Weekly snapshot
 - Daily snapshot
 - Synchronous
 - We later found out that our AIX administrator erroneously put everything into one volume group, which replicated asynchronously
 - This has yet to be corrected because of extenuating circumstances
- Set up appropriate directory structures and permissions within the new filesystems for each database
 - We wrote an AIX shell script to do this

After we had determined our storage requirements for the six new filesystems, our next step was to physically allocate the space on the servers in scope.

On the AIX server, our AIX system administrator needed to allocate three separate volume groups, each backed by a set of one or more separate LUN's (logical unit numbers) on the EMC SAN. We needed three separate volume groups because we had three separate sets of replication requirements for our DB2 filesystems: weekly snapshot replication, daily snapshot replication, and synchronous replication. The assignment of filesystems to volume groups was documented in an earlier page of notes in this presentation.

An aside: a couple of months after our DR solution for DB2 went into production, we inadvertently discovered that our AIX system administrator who set up the volume groups and filesystems had erroneously defined all six filesystems into a single volume group, which was being replicated asynchronously between our production and DR data centers. While this did not cause any problems during our testing or our day-to-day operations, we flagged this as something to be fixed at the next convenient opportunity. However, we have not yet had the opportunity to correct this deviation from our design due to extenuating circumstances, which will be described in a later slide.

Once the volume groups and filesystems were created on the AIX servers, the next step was to create the appropriate directory structures and set appropriate file permissions on those directories. A simple parameterized AIX shell script was written to facilitate this.

Challenges Faced and Successes

- Change NEWLOGPATH database configuration parameter to move the active logs to the /db2activelogsn/dbname filesystem at next startup of database
- Compile and install new db2uext2 exit for archive logs
 - Using sqllib/samples/c/db2uext2.cdisk as source
- Stop and restart the DB2 instance
- Full offline backup of database, followed by redirected restore of the database into the new filesystems
 - /db2tablespacen/dbname
 - /db2indexspacen/dbname
 - /db2tempspacen/dbname
- Change backups to use new backup/cleanup script

27



Once all the physical storage was in place, the only thing left to do was to actually move the databases in scope to these new filesystems. This turned out to be a straightforward process, although it did require a brief outage of the databases while the moves were being done.

The first thing that we did was update the database configuration to change the NEWLOGPATH parameter to point to the /db2activelogsn/dbname filesystem and directory at the next startup of the database. This took care of populating the active logs filesystem.

Next, we compiled the new version of the db2uext2 user exit in order to change the destination of the archive log process from TSM to the /db2archivelogsn/dbname/NODE0000 filesystem and directory. Using the “db2uext2.cdisk” program that can be found in sqllib/samples/c as source, we had to change only a couple of lines in the program to fit our requirements.

Next, we had to stop and restart the DB2 instance for the databases in scope in order for the new archive log user exit to take effect.

The last step of the database migration process was to take a full offline backup of the database, and then perform a redirected restore of the database into the new filesystems. This would populate the /db2tablespacen/dbname, /db2indexspacen/dbname, and /db2tempspacen/dbname filesystems and directories.

After these steps were done, the only remaining step was to change the backup job for the database to use the new backup/cleanup script in order to start taking backups to the /db2backupsn/dbname/current directory and filesystem.

This completed the actual migration of databases into the new disk architecture. All that needed to be done was Disaster Recovery testing, to validate the process from end-to-end.

Challenges Faced and Successes

- First database was moved to the new disk architecture in January 2004
- Database was successfully recovered during first Disaster Recovery test for AIX in March 2004
- Three additional databases moved to the new disk architecture in August 2004

PNX has had a mature DR plan in place for our mainframe platform for almost 15 years, and a “quick and dirty” DR plan for a small portion of our “Wintel” platform since 2001. However, we did not have a comprehensive enterprise wide Business Continuity and Disaster Recovery solution in place until recently.

PNX’s Disaster Recovery project for our distributed platforms started as the IT component of our enterprise level Business Continuity initiative in 2003, and was completed in early 2004. The basic outline of the DB2 component of the AIX DR plan was drafted in late 2003, with the first database moved into the new disk architecture in January 2004. This first database was small in terms of size and transaction workload (about .5gb in size, and approximately 100 insert/update/delete transactions per day), but was deemed critical enough to the business to be needed quickly in the event of a disaster.

Our first DR test for the AIX platform was held in March, 2004 and was a complete success. Our recovery of the database in scope went flawlessly. The database was recovered up to the point of failure within 3 minutes of DBA participation in the test.

In August of 2004, three additional databases were moved into the new disk architecture for DR. The largest of these databases is approximately 7.5gb in size, with about 8mb of logged transactional activity on a daily basis.

PNX has not conducted any additional DR tests since the initial test in March 2004, for reasons that will be noted on the next slide.

What's Next?

- In July 2004, PNX outsourced our IT Infrastructure
 - As of May 2005, our production and DR data centers will be housed in vendor facilities in the Midwest, about 10 miles apart
 - DR is now solely supported by the outsourcing vendor
 - For now, PNX is still operating within this DR architecture
- Data Warehouse project, currently in development, may need to be moved into this DR architecture
 - Warehouse databases too large for full backups to disk on a regular basis
 - Tablespace level backups instead of database level backups
 - Incremental backups instead of full backups



In July 2004, PHX signed an contract to outsource our IT Infrastructure. As part of this agreement, the outsourcing vendor would assume responsibility for hosting both our production and DR data centers at their facilities. They would also assume support of the DR process itself for all computing platforms, including the AIX platform.

Between October 2004 and May 2005, our entire production and DR data centers in Connecticut and New York will have been migrated to vendor facilities in the Midwest. The production data centers (Enfield, and a small one in Hartford, CT) will be consolidated to one vendor data center, and the DR data centers (Albany, and a portion of Enfield) will be consolidated to a second vendor data center that is located about 10 miles away from the first one.

For the time being, until the outsourcing vendor proposes otherwise, we are still working within the framework of the AIX DR strategy that we developed and implemented in 2003/2004.

With that in mind, we are moving forward with planning to move additional applications into this DB2/AIX DR architecture as the business dictates. The biggest application on the horizon is a Data Warehouse application that is currently in development which the business is considering as a possible candidate for falling within the 72 hour window of recoverability. While details about the size and scope of the Data Warehouse are still in the analysis and requirements gathering phase, it is anticipated that the underlying DB2 databases will be too large for creating full database backups that reside on disk on a regular basis. We will also be considering the need to perform tablespace level backups instead of database level backups, and using incremental backups as opposed to full backups for the Warehouse. This poses a number of interesting challenges in that we will need to incorporate these new backup requirements into our DB2/AIX DR strategy.

What's Next?

- What would have been next if we had not outsourced?
 - Assessment and impact of new DB2 features introduced in v8.1/v8.2
 - Infinite logging
 - Autonomic log manager
 - Can use new DB CFG parameters as an alternative to the old “dsnuext2” userexit
 - Can identify two target destinations for archive logs (e.g. TSM and disk)
 - Logs stored in online backup images
 - All logs needed to restore the online backup and roll forward to the time corresponding to end of the backup can optionally be stored in the backup image
 - Restore of database can also optionally restore the logs

Had we not outsourced our IT Infrastructure, we would have upgraded our DB2/AIX environment to v8 by now, and would have had a number of new features to assess in terms of how they impacted our DR strategy for DB2. This would include items such as:

- Infinite logging - Introduced in DB2 v8.1, infinite logging is enabled by setting the LOGSECOND parameter to -1. When this is enabled, the number of active logs that can be used by a database is no longer constrained by the sum of (LOGPRIMARY + LOGSECOND). If a long running unit of work (or a number of concurrent long running UOW's) use up all the active logs as specified by LOGPRIMARY, DB2 will invoke the userexit to archive logs that are still active in order to allow the active UOW's to continue processing. This means that the active log directory may not always contain all the active logs. In the event of a rollback or crash recovery scenario, the userexit may be invoked to retrieve some of the archived active logs in order to complete processing. This could have a severe impact to rollback or crash recovery processing.
- Autonomic log manager - The log manager was rewritten in DB2 v8.2. Some of the new options and features that are included in the new log manager is the ability to use new database configuration parameters to control archive log processing as an alternative to the old “dsnuext2” userexit (although the userexit is still supported). Also, the new DB CFG parameters will allow up to two target destinations for archive logs. For example, you can specify that a log can be archived to both disk and TSM at the same time, if desired.
- Logs stored in online backup images - Also introduced in DB2 v8.2, you now have the ability to optionally include all the log files required to restore and roll forward a database to the time corresponding to the end of the backup within the backup image itself. You also have the option of restoring the log files from the backup image while restoring the database, or just restoring the log files themselves if desired.

References

- IBM Training Course
 - “DB2 UDB Advanced Recovery for Single Partitioned Databases”, CF491

I attended a session of the IBM “DB2 UDB Advanced Recovery for Single Partitioned Databases” training class (IBM course code CF491) in New York City in August of 2003, and I found this class to be the single most informative class that I have taken in my IT career. Much of the foundation of the DB2/AIX Disaster Recovery solution detailed in this presentation came from information and knowledge that I gained from attending this class. The student handbook that was used in the class is one of the more valuable technical references that I use on a regular basis.

I would highly recommend this class to anybody interested in gaining an in-depth knowledge of what’s going on “under the covers” when dealing with DB2 database backup and recovery.

*Disaster Recovery and DB2/AIX: A User Experience
Session: D10*

Bill Gallagher
Phoenix Life Insurance
bill.gallagher@phoenixwm.com



I hope you found this presentation of value. Please feel free to contact me at the email address above should you have any questions or would like any additional information about our Disaster Recovery solution for DB2/AIX.