

2-6 October, 2006

Hilton Vienna

Vienna, Austria

IDUG® 2006
Europe

Session G07

Get Up: DB2 High Availability & Disaster Recovery

Robert Catterall
CheckFree

Tuesday, October 3, 2006 • 3:15 p.m. – 4:15 p.m.

Platform: Cross-Platform

GoFurther



Abstract:

Bad things can happen to good systems. Servers can fail, as can disk arrays, operating systems - even DB2 itself. Sometimes, an entire data center becomes inoperative. Fortunately, there are many things you can do to help ensure that your DB2 systems will be highly available and capable of surviving disaster-scope events. This session will cover several of these protective measures from a cross-platform perspective.

Author:

Robert Catterall
Director, Engineering
IT Strategy, Planning & Architecture
CheckFree

rcatterall@checkfree.com

Agenda

- Availability and recovery: fundamentals and lingo
- DB2 for z/OS data sharing
- Traditional DB2 for Linux/UNIX/Windows (LUW) failover clustering
- DB2 for LUW HADR
- Two- and three-site DR configurations

2

- I. Availability and recovery: fundamentals and lingo
 - A. Targets: RTO and RPO
 - B. Localized failures and disaster-scope events
- II. DB2 for z/OS data sharing
 - A. Technology overview
 - B. Planned and unplanned outages
- III. Traditional DB2 for LUW failover clustering
 - A. Technology overview
 - B. Implementation options
- IV. DB2 for LUW HADR
 - A. Relationship to log shipping
 - B. Maintenance without the window
- V. Two- and three-site DR configurations
 - A. When distance is a requirement
 - B. Synchronous data replication to a nearby high availability site
 - C. The “hot-hot” option

About CheckFree



- Headquarters: Atlanta, Georgia, USA
- Main line of business: internet-based electronic billing and payment (EBP) services
- DB2 platforms:
 - Primary online transaction processing (OLTP) system: DB2 for z/OS V7 in data sharing mode on 3-mainframe parallel sysplex
 - Operational data store: DB2 for z/OS V7 system
 - Enterprise data warehouse: DB2 for AIX V8.2, with data partitioning feature (DPF) – 8 logical nodes on 2 pSeries servers
 - PeopleSoft CRM application: DB2 for Solaris V8.2

3

CheckFree is headquartered in the Atlanta area in the United States.

Our main line of business is electronic billing and payment, or EBP. Most of the end users of our system get to us via a Web browser through their bank's online banking application.

We run DB2 on several platforms:

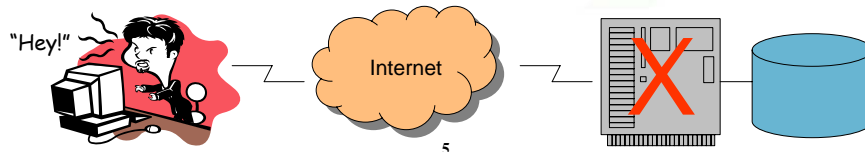
- Our primary online transaction processing (OLTP) system uses DB2 for z/OS V7, running in data sharing mode on a 3-mainframe parallel sysplex (there are nine members of the data sharing group - 3 on each mainframe). The size of this database is approximately 5 TB.
- We have an operational data store (ODS) which uses DB2 for z/OS running in standalone mode (i.e., non-data sharing). The size of this database is about 2 TB.
- Our enterprise data warehouse (EDW) uses DB2 for AIX V8.2, with the data partitioning feature (DPF - formerly known as DB2 EEE). This system has 8 logical DB2 nodes on 2 IBM pSeries servers. The size of this database is about 2.5 TB.
- We run the PeopleSoft CRM application, and that system uses a database managed by DB2 for Solaris V8.2. The size of this database is about 1.5 TB.

Fundamentals and Lingo

Following are some slides that address fundamentals of DB2 high availability and disaster recovery, along with related lingo (terms and acronyms).

Be prepared for: localized failures

- Examples:
 - Database server hardware failure
 - Operating system crash
 - DB2 failure (even the best DBMS will fail on occasion)
 - Disk subsystem failure
- ITIL (IT Infrastructure Library): Availability Management
- Server clustering is generally your best defense against database server hardware or software failure
 - We'll look at options for z/OS and Linux/UNIX/Windows servers



By “localized failures,” I mean failures that generally wouldn't be thought of as disasters, if by disasters we mean problems that result in the functional loss of a data center. Examples of localized failures include database server hardware failures, operating system crashes, DB2 failures (yes, even DB2 can fail), and disk subsystem failures.

Is your organization getting into the study and application of ITIL, the IT Infrastructure Library (a set of processes and best practices around IT management)? We are. The area of ITIL known as Availability Management is focused on (among other things) the minimization of downtime due to localized failure events.

Generally speaking, database server clustering provides your best defense against loss of data access due to localized failure events. In subsequent sections of this presentation, I will cover server clustering solutions for z/OS and Linux/UNIX/Windows platforms.

What about disk subsystem failures?

- Important: RAID technology reduces – but does not eliminate – the risk of a disk subsystem failure
- If your disk subsystem does fail, you might need to run the RECOVER utility to get back what you've lost
 - So, you still need to regularly back up your database objects
 - Backup frequency varies by installation, but it is fairly common to:
 - Perform a full backup of a database object once per week
 - Run incremental backup (capture changes since the last full backup) once per day between the weekly full backups
 - Keep the backup files for a couple of weeks before deleting them
 - For faster log apply, keep at least the most recent 24 hours of data changes in the active log files (as opposed to the archive log files)

6

Do you think that placing your DB2 database on RAID-type disk devices (i.e., devices that utilize redundant arrays of independent disks) provides complete protection from disk-related failures? Believe me, it doesn't. So, you could face a situation in which you need to run the DB2 RECOVER utility to restore a database object lost due to a failure in the disk subsystem. That being the case, you need to periodically create the backup copies of database objects that are an input (along with transaction log file data) to the RECOVER utility.

Database backup frequency will vary from site to site, but it's not uncommon for people to create a full backup of an object once a week, with incremental backups (which capture changes made since the most recent backup) run once per day between the weekly full backups. Retain backup files for a while before discarding them (two weeks might be a reasonable retention period). Keep in mind that for optimal performance, you want log records to be applied from the active log files (that is, not the archive log files), so you should probably plan on having at least the most recent 24 hours of database change activity recorded in the active log files. This is a matter of providing sufficient disk space, based on the level of data change activity on your system.

So, you have the database backup files

- But when will you use them?
- In case of a disk subsystem failure?
 - Maybe not
 - Suppose you have implemented a data change replication solution that lets you switch to a backup system (with a complete and current copy of the database) in case of a primary system failure
 - If you could switch to that backup system in less time than it would take you to recover the objects stored on a failed disk subsystem – wouldn't you?
 - We'll cover some of these replication solutions momentarily
 - *And you still need to backup your database – it's the ultimate insurance against data loss (especially if copies of backup files are stored off-site)*



7

Even if you backup database objects (fully or incrementally) on a daily basis, and you have the log records needed to complete a recovery operation available in the active log files on disk, it can take a while to recover a really big table or index. Well, suppose you didn't run RECOVER following the loss of an object due to a disk subsystem failure? Consider this alternative: you have a complete copy of your database at a disaster recovery (DR) site, and you keep that copy of the database in synch with the primary copy. If you could switch processing to that DR site in less time than it would take you to recover the lost database object using the RECOVER utility at the primary site, wouldn't it make sense to effect the site switch? I will describe multi-site high-availability configurations in the last section of this presentation.

Note: I'm not recommending that you stop backing up your DB2 database. ALWAYS do that, as it is the ultimate in data loss insurance (especially if the backup files are stored in a hardened off-site facility). I'm just suggesting that, depending on your configuration, you might not need to use those backup files to recover from certain outage situations.

Be prepared for: disaster-scope failures

- Definition: an event that necessitates the relocation of an application system from one data center to another
- Examples:
 - Fires
 - Floods
 - Storms
 - Earthquakes
 - Explosions
- Process whereby application service is restored following such an event is known as disaster recovery (DR)
- ITIL (see slide 5): IT Service Continuity Management



8

There are events that would necessitate the relocation of an application system and all that goes with it (including the database) from one data center to another. These events, which can include floods, fires, storms, earthquakes, and explosions, are called disasters in the context of application availability. The process of restoring application service following such an event is known as disaster recovery, or DR.

In the notes for slide 5, I mentioned that CheckFree, like many other organizations, is focused on the application of best practices as documented in ITIL, the IT Infrastructure Library. Within ITIL, a practice area known as IT Service Continuity Management is concerned with DR preparedness.

What are your DR objectives?

- Usually expressed as upper limits on:
 - Time to restore application service at the alternate data center
 - Called the recovery time objective (RTO)
 - Amount of data that will be lost if a disaster event occurs
 - Called the recovery point objective (RPO)
- Example: in the event of a disaster, XYZ Company wants application service restored within 2 hours, with no more than 15 minutes of data lost



9

Before you start evaluating DB2 DR options, decide what your recovery objectives will be. Generally speaking, these objectives are expressed as upper limits on 1) the amount of time it will take to restore application service at the alternate data center and 2) the amount of data that will be lost if a disaster event occurs. The former is called the recovery time objective (RTO); the latter is called the recovery point objective (RPO).

On this slide, I've provided a simple RTO/RPO example. Company XYZ has set DR goals as follows: if a disaster-scope event incapacitates the primary data center, application service will be restored at the alternate site within two hours (the RTO), and no more than 15 minutes of data will be lost (the RPO).

Note: RTO, RPO not just DR terms

- You can have RTO and RPO targets related to localized failure events, too
- At CheckFree, we preface RTO and RPO with:
 - “Operational” when referring to targets pertaining to localized failure events (ORTO, ORPO for short)
 - “Disaster” when referring to targets pertaining to disaster-scope events (DRTO, DRPO for short)
- Note that for localized failures, the RPO tends to be zero (no loss of committed DB2 database changes)
 - And so it should be, as long as the transaction log is preserved

10

Although the acronyms RTO and RPO are generally used in the context of DR planning, objectives related to service restoration and data loss can certainly be established for recovery actions related to localized failure events. At CheckFree, we distinguish between DR-related RTO and RPO targets and those related to localized failure events by prefixing the acronyms with “disaster” and “operational,” respectively. So, a service-restoration target related to DR is the disaster RTO, or DRTO, and a data loss minimization target related to localized failure events is an operational RPO, or ORPO.

I’ve observed that the RPO for localized failure events tends to be zero, meaning no loss of committed database changes. That may seem to be a pretty stringent goal, but in fact zero loss of committed data changes should be expected if the DB2 transaction log is not lost in the failure event. No database change is committed until it has been recorded in the log file.

What should RTO, RPO targets be?

- Different targets appropriate for different organizations
 - Factor: the environment in which the organization operates
 - What do customers expect/demand?
 - What are the DR capabilities of competitors?
 - Factor: costs associated with DR solutions
 - Generally speaking, the smaller your RTO and RPO targets, the more money it will take to achieve the objectives
 - Can you justify higher DR costs?



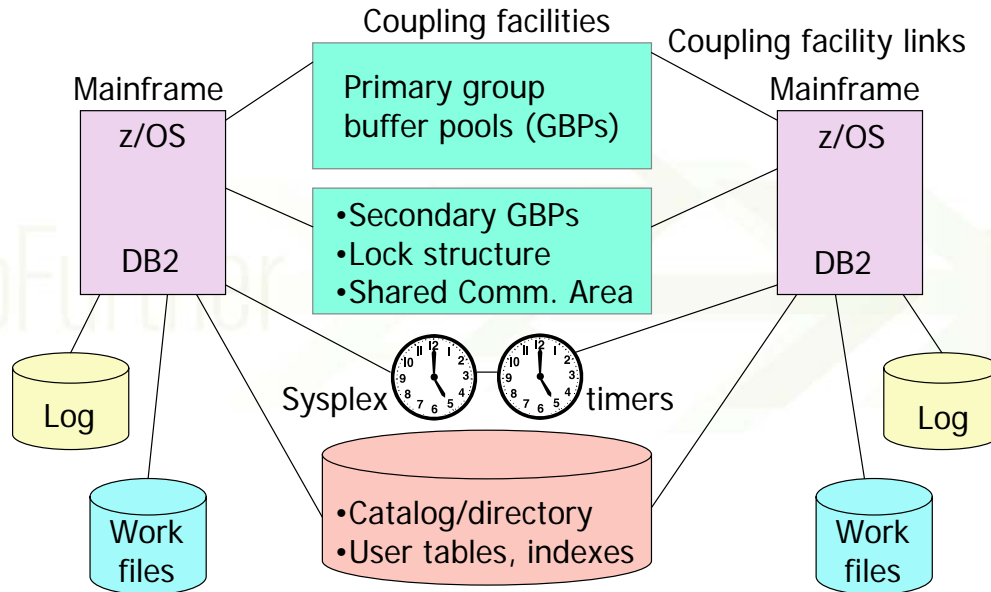
11

There are no universally correct RTO or RPO targets. Generally speaking, the smaller the numbers used to express an RTO and an RPO, the more money it will take to achieve these objectives, so an organization must balance the cost of a DR solution with the requirements of the marketplace in which the organization operates. A financial services company, for example, might decide that the investment required to achieve a one-hour RTO and a one-minute RPO (aggressive targets) is justifiable, given its customer demands and its competitors' capabilities. An organization in a different industry might decide that a lower-cost solution that delivers a four-hour RTO and a 30-minute RPO is the right choice.

DB2 for z/OS
Data Sharing

The server clustering technology that provides high availability for DB2 for z/OS is called DB2 data sharing.

DB2 Data Sharing in a Parallel Sysplex

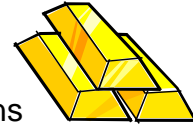


13

The diagram on this page shows the components of a DB2 data sharing group (ours has 9 DB2 subsystems running on 3 mainframes):

- **Mainframe computers** (a mainframe parallel sysplex can include up to 32 servers).
- **Coupling facilities:** Shared memory devices in which reside structures such as group buffer pools, the global lock structure, and a shared communications area (used to hold information about DB2 objects in an exception state). They provide super-quick access to information needed by members of the data sharing group.
- **Coupling facility links:** These provide very-high-bandwidth paths between servers in the parallel sysplex and the coupling facilities. Each link may handle several thousand requests per second.
- **Sysplex timers:** These devices (two, for redundancy) ensure that the servers in the sysplex all use a common clock for timestamp values.
- **Common database objects:** The DB2 subsystems which are members of the data sharing group share a common DB2 catalog and directory, and they are all co-owners of all the user tablespaces and indexes defined in the catalog.
- **Log data sets:** Each member of the data sharing group has sole write access to its log data sets, and read access to other members' log data sets (needed for recovery of objects updated by programs running on different members of the group).
- **Workfile data sets:** Each DB2 member has its own, and these are not shared.

Data sharing and localized failures



- The gold standard of high-availability solutions
 - Up to 32 DB2 subsystems (called members of the data sharing group) co-own a common database
 - All subsystems share concurrent read/write access to the database
 - No “master” subsystem – all are peers
 - Lose a subsystem (or mainframe or z/OS image), and there is no wait for another DB2 system to take over for the failed member
 - No wait because there is no takeover – the other DB2 members already have access to the database
 - Application workload shifts from the failed member(s) to the surviving member(s)

14

I think of DB2 data sharing as the gold standard of high-availability solutions. In a data sharing system, up to 32 DB2 for z/OS subsystems co-own a single database. These subsystems (called members of the data sharing group) share concurrent read/write access to the database, and none of the members has any kind of special status relative to the others; that is to say, there is no “master” subsystem, only peer members. If a mainframe server or a z/OS operating system image or a DB2 subsystem in the sysplex fails, there's no delay while another DB2 subsystem takes over for the failed member. That's because there is no take-over—the other members in the data sharing group already have access to the database. The system will automatically redistribute application work from the failed subsystem to other members of the data sharing group.

Is failure impact completely eliminated?

- Not quite
 - Database pages X-locked (being changed) by DB2 member at time of failure remain locked until failed member is restarted
 - Mitigating factors:
 - Given reasonable commit frequency, relatively few pages will be X-locked by a DB2 member at any given time
 - Failed member can be automatically restarted (in place or on another z/OS image in the sysplex)
 - Restart is fast (less than 2 minutes in our environment)
- Bottom line: impact of DB2 failure greatly reduced in a data sharing system

15

Is a component failure in a parallel sysplex a completely non-disruptive event? No. Database pages X-locked (that is, in the process of being changed) by a DB2 member at the time of that member's failure (due to hardware failure, operating system failure, or the failure of DB2 itself) remain locked until the failed subsystem is restarted. Typically, the failed DB2 member will be automatically restarted (in place, or on another z/OS image in the sysplex if the system on which the member had been running is not available), and the restart can complete fairly quickly (we've observed restart times under 2 minutes here at CheckFree), so the retained locks will not be held for long. While the failed DB2 member is restarting, pages not locked by the failing subsystem remain fully available for read/write access.

Not just a defense against failures

- Data sharing also great for eliminating planned outages
- Example: apply DB2 (or z/OS) software patches (fixes)
 1. Apply maintenance to load library (DB2 binaries)
 2. Quiesce application traffic on a DB2 member (allow in-flight work to complete), and spread workload across remaining members
 3. Stop and restart quiesced DB2 member to activate patches
 4. Resume the flow of application traffic to the DB2 member
 5. Repeat steps for remaining members of the data sharing group
- Result: maintenance without the maintenance window
- Maintenance requiring group-wide shutdown very rare

16

DB2 for z/OS data sharing is also a great solution for eliminating many planned outages. Our data sharing group enables us to apply DB2 (or z/OS) software patches in such a way that application access to the database is not affected: We simply quiesce traffic on a member (allowing in-flight units of work to complete), spread the application workload across the remaining members, shut down and restart the quiesced subsystem (thereby activating the previously applied software patch), and resume the flow of traffic to that member. We repeat these steps in a round-robin fashion until the new software maintenance is active on all members of the data sharing group – all with no disruption of the application workload.

This “window-free” scheme of maintenance application is possible because IBM tests different maintenance levels to ensure that DB2 subsystems at maintenance level n can coexist with subsystems at maintenance level $n+1$ while the software fixes are being applied in sequence to each member of the data sharing group, as described above. Maintenance requiring a group-wide shutdown is very rare (an example is a change to the global locking algorithm that requires all members to use the new algorithm at the same time – there have been very few such changes since data sharing was introduced).

Is the price right?



- Not free, but quite a bit less expensive than in years past
 - Internal coupling facilities
 - Reduced data sharing overhead (data sharing adds about 7.3% to the CPU cost of SQL statement execution in our environment – and that is with a high degree of inter-DB2 read/write interest)
 - Faster coupling facility processors
 - Faster coupling facility links
 - Code enhancements (DB2, z/OS, coupling facility control code)
- If you have thought before that DB2 data sharing is not right for you, it may be time to reconsider

17

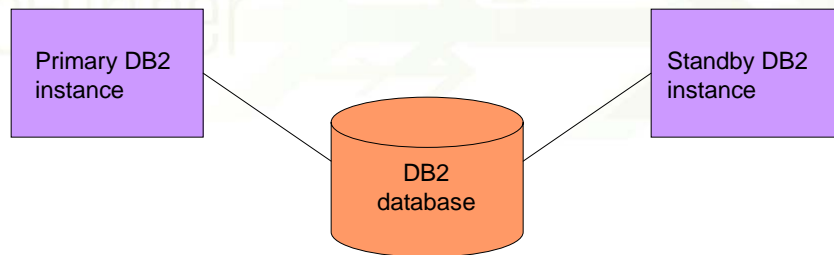
Some of you mainframe DB2 people may be working in an environment without a parallel sysplex because your organization's leaders think that DB2 data sharing is too costly an option. If so, take another look at this technology. The required infrastructure is considerably less expensive than it was several years ago, largely because coupling facilities can be LPARs within mainframe servers, whereas they were once available only as separate external devices. As for the overhead of DB2 data sharing, that, too, has decreased significantly over the years, thanks to faster coupling facility processors, faster coupling facility links, and enhancements made to coupling facility control code, DB2, and the z/OS operating system. At CheckFree, we have a very high degree of inter-DB2 read/write interest in our production data sharing environment (that is, most of the objects in our database are accessed in read/write mode from several member DB2 subsystems concurrently), and the level of data sharing overhead we see is around 7.3% (that is to say, an SQL statement executed in our data sharing group will consume about 7.3% more in-DB2 CPU time than it would if it were executed in a stand-alone DB2 for z/OS subsystem). We see 7.3% overhead as being a very reasonable price to pay for the availability (and scalability) advantages of data sharing.

**Traditional DB2 for
LUW Failover
Clustering**

Next, a look at traditional failover clustering technology for DB2 on Linux, UNIX, and Windows servers (LUW).

A tried-and-true high availability solution

- Not a shared data configuration, although both DB2 instances are physically connected to the database
- If the primary DB2 instance fails, the standby will automatically take over the database connection



19

Technology akin to DB2 for z/OS data sharing isn't available for DB2 for LUW systems, because DB2 on those platforms employs a "shared nothing" architecture, vs. the shared data architecture of DB2 on mainframe servers. Failure of a DB2 for LUW server means a temporary loss of access to all data managed by that server, until the DB2 instance can be restarted. The typical means of minimizing this restart time has been to deploy DB2 for LUW on a server that is configured as part of a failover cluster. In such a cluster, if the server on which DB2 is running fails, a standby instance on another server can automatically take over the database belonging to the failed DB2 instance.

Failover clustering in action

- If primary DB2 fails, database will be available as soon as standby DB2 completes rollback/roll forward processing
 - Back out database changes associated with in-flight units of work
 - Externalize committed data changes not yet written to disk
 - Elapsed time will vary based on workload and DB2 parameter settings, but usually not more than a minute or two
- This is what DB2 would do in a non-clustered configuration in the event of a loss and subsequent restoration of power.

20

In a DB2 for LUW failover cluster configuration, when the primary server fails the standby DB2 instance will make the database available again as soon as it can complete the rollback (to back out units of work that were in flight at the time of the failure) and roll forward (to externalize pages that had been updated in memory on the failed instance and not yet written to disk) needed to restore the database to a consistent state (this is what a non-clustered DB2 for LUW system would do if you were to power the server off with DB2 running and then power it back on).

The standby DB2 instance in a failover cluster can complete restart processing pretty quickly when a failover occurs (generally in a minute or two, depending on workload and DB2 parameter settings),

More on traditional failover clustering

- May use capability provided by operating system:
 - Example: HACMP (High Availability Cluster Multi-Processing) in an AIX environment
 - Example: MSCS (Microsoft Cluster Server) in a Windows environment
- Alternatively, may use failover clustering software provided by third-party vendors
- Data-requesting client applications typically can reconnect to DB2 data server using same IP address
 - In that sense, failover is transparent to data requesters

21

Sometimes a DB2 for LUW failover cluster is implemented using a capability provided by the operating system. Two examples of such capabilities are High Availability Cluster Multi-Processing, or HACMP, in an AIX environment; and the Microsoft Cluster Server (MSCS) feature available with some editions of the Windows operating system.

Another means of implementing a DB2 for LUW failover cluster is to use clustering software provided by a third-party vendor.

In any case, resumption of application processing is facilitated by the fact that an application server will typically connect to the standby DB2 server (following a DB2 failover) via the same IP address as was used to communicate with the primary DB2 instance prior to the failover to the standby server.

DB2 for LUW
HADR

DB2 for LUW failover clustering is a good high availability solution. DB2 HADR is a better solution.

High Availability Disaster Recovery

- Introduced with DB2 for LUW V8.2
- A huge advance in DB2 for LUW high availability capability
 - As with traditional failover clustering, can be used to recover from localized failures with no loss of committed data changes (i.e., RPO = zero)
 - The difference: recovery time is MUCH faster (with HADR, you can achieve an RTO of a few seconds)
 - HADR can also be an effective DB2 for LUW DR solution (more to come on this)



23

HADR – short for High Availability Disaster Recovery – was introduced with DB2 for LUW Version 8.2. It was a major step forward with respect to enhancing the availability of DB2 for LUW systems.

With traditional DB2 failover clustering, database access can be restored relatively quickly after a primary-server failure, with no loss of committed database changes (that is to say, such a configuration can enable the achievement of an RPO of zero). HADR can also deliver on an RPO of zero, with the added benefit of a data access restoration time that is much smaller than that provided by traditional failover clustering technology: HADR can enable the achievement of an RTO that is expressed as a small number of seconds.

HADR can also be used as an effective DB2 for LUW DR solution. DR will be covered in a subsequent section of this presentation.

Think about DB2 for LUW log shipping

- A longtime means of keeping a backup copy of a DB2 database close to currency with respect to primary copy
- Start by backing up primary database and restoring it on standby server, so that two copies of the database exist
 - As data in the primary copy of the database is changed, the change actions are recorded in the DB2 transaction log
 - The inactive log files on the primary system are periodically backed up and sent to the standby server
 - Backing up active log files is not so good for performance
 - The log file backups are processed by the standby DB2 system to bring that copy of the database closer to currency

24

To gain an understanding of DB2 HADR, think about a DB2 for LUW feature called log shipping, used for years at many sites to keep a backup copy of a database close to currency with respect to the primary database. The concept of log shipping is pretty simple: first, you back up the primary DB2 database and restore the backup on a standby DB2 system, so that two copies of the database exist. Following the database backup and restore operation, data changes (updates, inserts, and deletes) are processed on the primary DB2 server and recorded in the transaction log of the primary DB2 system. The inactive log files of the primary system are periodically backed up (backing up the active log files is not good for system performance), and a copy of the log backup is processed by the standby DB2 instance to apply the changes recorded therein to the copy of the database on that system. In this way, the standby system's database is brought that much closer to currency with respect to the database associated with the primary server.

HADR: like extreme log shipping

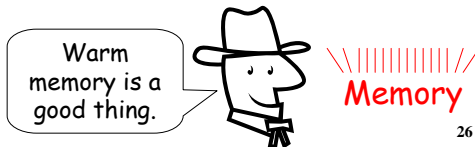
- As though the primary system's transaction log were backed up and transmitted to the standby system (and processed there) *for each record written to the log*
- Result: you can achieve much smaller RPO (even zero) versus log shipping, much smaller RTO versus traditional failover clustering



Conceptually, HADR is log shipping taken to the theoretical extreme, as though the transaction log at the primary site were backed up and transmitted to the DR site *for each record written to the log*; thus, HADR enables the achievement of aggressive RPO targets (even 0) that would not be possible using log shipping. On top of that, HADR dramatically speeds up service restoration time (the RTO target) versus traditional DB2 for LUW failover clustering.

How HADR speeds recovery time

- Memory of the standby DB2 system is kept current with respect to the primary system (in terms of data changes)
 - In other words, an updated page in the buffer pool of the primary DB2 will also be in the buffer pool of the standby DB2 system
 - Result: when it is time to switch systems:
 - Data change redo processing is unnecessary (data changes not externalized on the primary DB2 are in memory on the standby)
 - Rollback processing is minimized (pages targeted by undo processing are probably already in the standby DB2's buffer pool)



Here is how HADR speeds up data access restoration following a primary-server failure: because the standby DB2 instance is processing log records generated by the primary instance in real time, it keeps its memory (e.g., buffer pools) in a current state relative to the primary instance.

Why is this aspect of HADR – sometimes referred to as “warm memory” on the standby instance – important when it comes to speed of database service restoration? The twofold answer: first, with HADR, roll-forward processing (also known as re-do processing) is not needed, because pages that had been updated on the primary DB2 system and not yet written to disk are in memory on the standby DB2 system. Second, the most time-consuming task associated with rollback processing (rollback is the un-doing of changes made by in-flight units of work) in normal DB2 restart and recovery – the random I/O operations needed to bring affected pages into the buffer pool – is unnecessary in an HADR configuration, because those recently updated pages are probably already in the buffer pool of the standby DB2 instance. HADR virtually eliminates random I/O during a DB2 system switch-over.



Three flavors of HADR

- Synchronous: do not commit data change on primary DB2 instance until log records written to log file of standby DB2
- Near-synchronous: do not commit change on primary DB2 until log records written to memory of standby DB2
 - Near-synchronous is good for protection against localized failures
 - Less performance impact versus synchronous
 - No loss of committed data changes unless you simultaneously lose primary and standby DB2 instances (highly unlikely)
- Asynchronous: committing of changes on primary DB2 does not wait on communication of changes to standby
 - Can be a very good DR solution (more to come on this)

27

There are three flavors of HADR:

- *Synchronous* – A database change made on the primary DB2 instance will not be committed on that instance until the associated log record (or records) has been written to the transaction log file on the standby DB2.
- *Near-synchronous* – Same as synchronous, except that the change made to the primary DB2 database can be committed when the associated log record (or records) has been written to memory on the standby DB2 instance (vs. written to the standby's transaction log).
- *Asynchronous* – The committing of a database change on primary DB2 for LUW instance is not held up in any way by the communication of log records to the standby instance.

Near-synchronous mode is an attractive solution for localized failure protection: it has less of a performance impact on the primary DB2 instance (versus synchronous mode), and if the primary instance fails you won't lose any committed data changes unless you simultaneously lose the secondary DB2 system in the HADR configuration (highly unlikely). Asynchronous mode could be a very good option for keeping a secondary DB2 instance located at a distant DR site very close to currency with respect to the primary instance (more on DR to come).

Reducing planned downtime with HADR

- High-availability application/activation of software patch:
 1. Temporarily stop the flow of log records from the primary to the standby DB2 instance in the HADR configuration
 2. Apply patch to the standby DB2 instance, and stop and restart that instance to activate the maintenance
 3. Resume transmission of log records to standby instance, and let that instance catch up and resynchronize with primary instance
 4. Initiate switch to make the former standby the primary instance (should take around 10 seconds)
 5. Repeat steps 1-3 to apply and activate the patch on the new standby instance (formerly the primary instance)

28

HADR does more than reduce the impact of unplanned outages. You can also use this DB2 for LUW feature to virtually eliminate planned downtime that otherwise would be needed to apply and activate software or hardware maintenance. The steps on this slide show how that could be done.

Really windowless maintenance?

- Yes, even with the database being inaccessible for 10 seconds or so (to accomplish the primary switch-over)
 - Downtime defined in terms of failed customer interactions (FCIs)
 - The database can be offline for 10 seconds without causing any FCIs (just slightly elongated response time for some transactions)
- Windowless maintenance formerly considered possible only via shared-data architecture (such as DB2 for z/OS data sharing)
 - HADR brings this capability to DB2 for LUW (though not yet for data partitioning feature)

29



Now, you might say that the approach, described on the preceding slide, to minimizing downtime related to the application of software maintenance in a DB2 HADR configuration is just that: a means of minimizing – versus eliminating – planned downtime. If it does entail a period of database inaccessibility – even a very short period (10 seconds or so) – can it be referred to as a way to apply maintenance without the need for a maintenance window? I would say yes, because I believe that database availability should be measured in terms of failed customer interactions (FCIs), and not just in terms of total uptime as a percentage of the total time in a given period. If the database is inaccessible for about 10 seconds during the switchover from one DB2 system in the HADR configuration to the other, will that cause any FCIs? If the switchover is performed during a period of relatively low application activity, I believe that the answer could be no – the switchover would not result in FCIs (some users might see temporarily elongated response time, but I would not expect timeouts). So, I do see HADR as a means of accomplishing maintenance application and activation without a maintenance window, and I no longer think of windowless maintenance as being something that can only be accomplished with a shared-data cluster (such as a DB2 for z/OS data sharing group).

Two- and Three-Site DR Configurations

Last but not least, a look at various DB2 DR configurations.

Do you need a far-away DR site?

- In the USA, you do
 - Earthquakes, hurricanes, tornadoes
 - One disaster event could incapacitate both the primary site and the DR site if they are too close together
- If Europe, maybe not
 - Much-reduced risk of natural disasters

Good



31

OK?



In the United States, if you have a 2-site DR configuration, it is a very good idea to have a good bit of distance between the primary and DR sites. If the two sites are separated by only a few kilometers, they could both be incapacitated by a region-wide disaster-scope event (e.g., an earthquake, a catastrophic flood, or even a tornado if both sites are in the path of the storm).

In most of Europe, natural disasters occur with much less frequency than is the case in most of the USA, so locating a DR site within a few kilometers of the primary data center may not be a risky decision.

Data to distant DR site: DB2 for LUW

- Old way: log shipping (see slide 24)
- Better way: HADR running in asynchronous mode
 - RPO: seconds of data loss, versus minutes
 - Synchronous or near-synchronous mode would probably result in unacceptable performance degradation for high-volume workload
 - Distant DR site = too much propagation delay
- Any reason to use log shipping vs. HADR?
 - Here is one: HADR is a point-to-point solution
 - Log shipping could be used in a three-site configuration (site A to site B and site A to site C)
 - Even with a three-site configuration, log shipping would not be my first choice (explanation forthcoming)

32

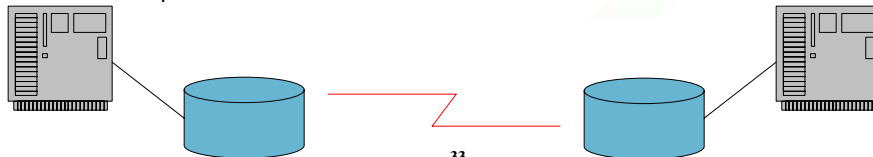
If you do have a DR site that is located hundreds of kilometers from the primary data center, how do you get data to the DR site to prepare for a possible DR situation? Consider this from a DB2 for LUW perspective.

One solution is log shipping, described on slide 24. A better solution would be to run HADR in asynchronous mode. This would enable you to hit much more aggressive targets for RTO and RPO, versus the log shipping approach. Why asynchronous? It's a matter of distance. Running HADR in synchronous or near-synchronous mode would probably have an unacceptably negative impact on the performance of the primary DB2 server, due to the propagation delay associated with transmitting log records over a distance of several hundred kilometers (assuming a far-away DR site). HADR running in asynchronous mode can keep the copy of the database at a distant DR site within seconds of currency relative to the primary-site database.

Log shipping MIGHT be seen by some as being preferable to HADR if there is a requirement that data be sent from the primary to more than one alternative site (HADR is a point-to-point solution); however, given such a requirement, log shipping would STILL not be my preferred solution, as I'll explain momentarily.

DB2 for LUW data to distant DR site...

- Another option: disk array-based asynchronous replication
 - Advantages:
 - Changes to any data (not just data in the DB2 database) are propagated to DR site (think about flat files, program libraries)
 - Very lean and mean, performance-wise (no need to go through all the layers of the TCP/IP stack – OS not involved)
 - Disadvantages:
 - Probably costs more than HADR
 - Requires same vendor's disk hardware at both ends



Another means of getting DB2 for LUW data to a distant DR site is disk array-based replication. One of the advantages of this solution is that it gets changes to anything in the disk subsystem – not just the DB2 database – to the DR site. So, no worries about how you'll keep program libraries (among other things) at the two sites in synch. Disk array-based replication is also very CPU-efficient, as communication is between the disk subsystems at the two sites and there is no need to go through the database server's operating system and all the layers of the TCP/IP stack.

Disk array-based replication is not without disadvantages. For one thing, it probably requires a greater investment of money than the DB2-native HADR solution. Additionally, the disk-based solution requires you to have the same vendor's storage hardware installed at both the primary and DR sites, and this loss of configuration flexibility can end up increasing your hardware costs.

Also, software-based replication

- Third-party host mirroring software
 - Interfaces with the server's file system
 - Every write I/O is sent simultaneously to disk at both sites



34

Yet another way to get DB2 for LUW data from a primary site to a distant DR site is to utilize a software-based data replication solution. An example of such a solution is a host mirroring software product. A product of this type interfaces with the database server's file system, and causes each write I/O operation to be driven simultaneously to both the local disk subsystem and the disk subsystem at the distant DR site.

Host mirroring: pros and cons

- Advantages:
 - Like disk array-based solution, propagates everything (not just database changes)
 - Can have different vendors' disk arrays at different sites
- Disadvantages
 - A little extra load on the server versus disk array-based replication
 - Probably costs more than HADR

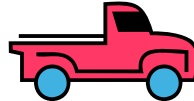
35

Like the disk array-based replication solution described on slide 33, host mirroring software causes changes made to any data at the local site to be mirrored at the DR site – not just changes made to data in the DB2 database. Unlike the disk array-based solution, host mirroring software does not require that the same vendor's storage hardware be installed at both the primary and the DR sites.

Compared with disk array-based replication, host mirroring software puts a little extra load on the primary-site database server, as each write I/O operation has to be driven twice (once to the primary site and once to the DR site). Additionally, this solution tends to cost more to implement than does the HADR feature of DB2 for LUW.

Data to distant DR site: DB2 for z/OS

- Old way: “pick-up truck methodology” (JR Brown, IBM)
 - 1 or 2 times per day, load DB2 archive log and image copy tapes onto a truck, which then delivers them to DR site
 - In the event of a disaster:
 - IPL mainframe at DR site
 - Perform DB2 conditional restart
 - Use RECOVER utility to restore tablespaces from image copies and get them as close to currency as possible using archive log files
 - Problems:
 - Might take several days to restore application service after a disaster
 - Data loss could exceed 24 hours



36

How about getting DB2 for z/OS data to a distant DR site? It wasn't too long ago that the predominant means of doing this was what Judy Ruby-Brown, my friend and longtime colleague at the IBM Dallas Systems Center, called the “pick-up truck” methodology: once or twice a day, you took your DB2 archive log and image copy tapes, loaded them in a truck and had them delivered to your DR site. In the event of a disaster, you IPLed (for distributed systems folks: booted up) your mainframe at the DR site, did a conditional restart of the DB2 system (lots of fun), and used the RECOVER utility to restore all the tablespaces from the image copies and get them as close to currency as possible using the archive log files. Using this process, it could easily take two or three or more days to restore service for a large application system following a primary site disaster event, and the amount of lost data could be 24 hours or more. Yuck.

Bye-bye truck, hello disk

- Availability of disk array-based replication technology (see slide 33) was even better news for mainframe folks than it was for users of Linux/UNIX/Windows servers
 - At least the DB2 for LUW users had log shipping
 - Array-based replication once mainframe-only (though not now)
- With asynchronous array-based replication, data loss (due to disaster) went from hours (or days) to seconds
 - Another plus: database recovery at DR site much simpler, faster
 - Like recovering from a local power outage: START DB2
 - Made possible through mirroring of active log data sets at DR site
 - Database ready for access in an hour or two, versus days before

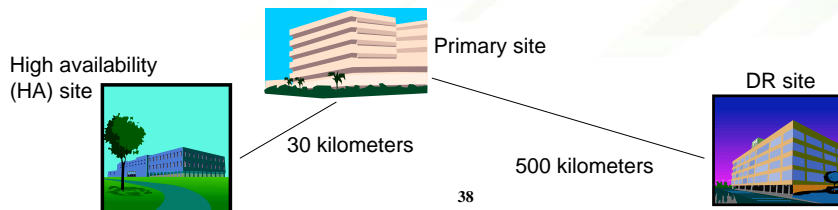
37

The pick-up truck methodology was made obsolete by disk array-based replication technology (described on slide 33), which got an even better reception at DB2 for z/OS sites than it did at DB2 for LUW sites (at least the DB2 for LUW folks had the log shipping option – generally a much preferred alternative to the pick-up truck). In fact, array-based replication was initially only available with disk subsystems that attached to mainframe servers (though that is no longer the case).

Instead of settling for a recovery time measured in days and data loss running into hours (or perhaps even a day or two), organizations using array-based replication in asynchronous mode could get a remote copy of a DB2 for z/OS database at a distant DR site ready for application requests in less than one hour, with only seconds of data loss. Additionally, array-based replication made recovery of the DB2 for z/OS database at the DR site much simpler versus the pick-up truck methodology: the process isn't unlike recovering locally from a power failure – it amounts to issuing the START DB2 command (it is this easy because the DB2 active log data sets are mirrored at the DR site, along with the database itself).

When are two sites not enough?

- Answer: when 1) RPO is zero, and 2) a configuration with only two sites, close together, is deemed too risky
- If RPO = 0, data replication must be synchronous
 - My opinion: not feasible if sites are more than about 40 fiber kilometers apart (usually about 25-30 straight-line kilometers)
 - But in some parts of the world, a 3rd site, located hundreds of kilometers away, would be needed to provide protection from a regional disaster



Some organizations spend the additional money needed to implement a DR configuration across three sites instead of two, and the motivation to do this generally involves two requirements: 1) that no committed database changes be lost if the primary site becomes inoperable, and 2) that at least one data center be located hundreds of kilometers from the primary site, due to the risk of large-scale disaster events.

The zero RPO target generally necessitates a nearby recovery site because it requires synchronous replication of database changes to the alternate site, and synchronous replication is usually not feasible if the target site is more than about 40 fiber kilometers from the source site (this typically translates into a straight-line distance of 25-30 KM, as it is unlikely that a fiber link will run in a straight line between the sites). The third site in a 3-site configuration is the distant DR site, usually located several hundred kilometers from the primary site (see slide 31). A distant DR site would be needed if having only two close-proximity sites would expose the organization to too much of a risk with respect to region-wide disasters. Note that in a three-site configuration, the nearby alternate site is sometimes called the high availability, or HA site. The distant site is referred to as the DR site.

Data to nearby high availability (HA) site

- DB2 for LUW
 - HADR in synchronous or near-synchronous mode (see slide 27)
 - Software-based host mirroring, synchronous mode (see slide 34)
- DB2 for LUW and DB2 for z/OS
 - Disk array-based synchronous replication
- These options can be used in combination with various asynchronous replication technologies (slides 32-35, 37)
 - Example:
 - Replication to HA site: synchronous disk array-based for mainframe servers, software-based host mirroring (synchronous) for LUW
 - Replication to distant DR site: asynchronous disk array-based for mainframe servers, HADR (asynchronous) for DB2 for LUW

39

There are several options available for getting data changes to a nearby HA site in a synchronous fashion. If a database is managed via DB2 for LUW, the HADR feature can be configured to run in synchronous or near-synchronous mode (see slide 27). Alternatively, the software-based host mirroring solution described on slide 34 can be set up to operate in synchronous mode. For both DB2 for LUW- and DB2 for z/OS-managed databases, disk array-based replication operating in synchronous mode can be a good solution.

In a three-site configuration, different synchronous data replication solutions can be paired with various asynchronous replication solutions to get data changes to the HA and DR sites. For example, an organization might decide to get DB2 for z/OS database changes to the nearby HA site using disk array-based synchronous replication, with software-based host mirroring running in synchronous mode being the preferred solution for DB2 for LUW systems. That same organization could use asynchronous disk array-based replication to get DB2 for z/OS database changes to the distant DR site, and HADR running in asynchronous mode to do the same for changes made to a DB2 for LUW database.

A DB2 data sharing consideration

- If running DB2 for z/OS in data sharing mode on a parallel sysplex, recovery time following a disaster will be somewhat longer than it would be for a standalone DB2
 - The reason: loss of the group buffer pools in the coupling facilities causes database objects to go into group buffer pool recover pending (GRECP) status
 - It takes a while to get the objects out of GRECP status when the data sharing group is recovered at the DR site
- If disaster looming, and you have time to initiate planned switch to HA site, you can significantly reduce recovery time (key is orderly shutdown of primary-site system)

40

Even if you synchronously mirror your DB2 database at a nearby DR site, your recovery time in the event of a disaster will likely be somewhat longer if you're running a DB2 for z/OS data sharing group as opposed to a standalone DB2 subsystem. The longer recovery time has to do with the loss of the group buffer pools when the disaster event occurs—something that causes database objects to go into group buffer pool recover pending (GRECP) status. It takes a while to get the objects out of GRECP status when the data sharing group is recovered at the DR site, and this delay is something that must be dealt with if a disaster event results in the sudden and complete failure of the primary-site data center. However, if a potential disaster event is looming and you have time (even just a few minutes) to do a planned switch from the primary site to a DR site, an IBM offering called a Geographically-Dispersed Parallel Sysplex (or Geoplex, for short) can significantly reduce the time required to make the data sharing group at the DR site ready for application activity.

Could you stretch your sysplex?

- Intriguing thought: spread DB2 systems in data sharing group across the two nearby sites in 3-site configuration
- Could this take disaster recovery time to just about zero?
 - Potentially, but only if you were to synchronously mirror coupling facility (CF) structures (e.g., lock structure, group buffer pools)
 - Problem: CF structures accessed very frequently (1000s of times per second), response time often only 20-30 microseconds
 - Given the speed of light, synchronously mirroring CF structures 30 KM apart would dramatically increase CF request response times
 - Stretched sysplex might be feasible if CFs about 1 kilometer apart



If we could figure out how to fold space...

41

Suppose you have a DB2 data sharing group running on a parallel sysplex. Given the availability advantages of such a configuration, you might think that a sysplex “stretched” over two sites separated by 20-30 kilometers (e.g., two mainframes in the sysplex at site A, and two at site B, with a coupling facility at each site) would provide even better protection against loss of application service; however, it turns out that a “stretched” sysplex would only provide super-fast recovery following a sudden and unplanned loss of one of the sites if the coupling facility (CF) structures (e.g., lock structure and group buffer pools) were synchronously mirrored between the two coupling facilities located 20-30 KM from each other. Synchronous mirroring over that distance is probably not feasible, because CF structures can be accessed more than 1000 times per second on a busy system, and average CF request response time is often in the 20-30 microsecond range. Even with the speed of light being what it is, the delay caused by synchronous mirroring of CF structure changes would dramatically increase CF request response times, and that would probably have a severe impact on system performance and throughput. It may be that a stretched sysplex configuration would perform acceptably well if the two parts of the sysplex were separated by a kilometer or less, but two sites located that close to each other could mean an increased risk of a double-site loss resulting from a natural disaster or another large-scale event.

The ultimate: never down, zero data loss

- Perhaps the only way to get there would be to eschew primary-backup configuration in favor of primary-primary
 - In other words, run a complete instance of the application system at more than one site, and split workload across the sites
 - If one site is incapacitated, don't recover it: just shift the workload that had been running at that site to the other site
 - Keep copies of database in synch or "near-synch" via bidirectional replication (near-synch assumes asynchronous replication)
 - May need to go with "near synch" if using software-based replication, but may need to go that route if record-level replication needed to reduce incidence of "collisions" (array-based replication is block-level)

42

Throughout this presentation, I've referred to primary and DR data centers. Suppose you have multiple data centers, with no one center having the primary designation. In other words, think of a situation in which you spread an application load across two or more sites, and in which your response to a disaster-scope event at one site is to simply redirect the traffic formerly sent to that site to the remaining sites. This approach is sometimes referred to as a "hot-hot" configuration because all sites take application traffic at all times during normal operations. In such a case, a full copy of the back-end database would likely have to be maintained at both sites, requiring bi-directional replication – most likely at a record level to minimize the possibility of "collisions." Note that record-level replication implies software-based replication, and such replication might need to run in asynchronous mode if the application workload volume is high; thus, the two copies of the database would be nearly in synch with each other, but not precisely in synch (there would be a slight time delay between the committing of a database change at site A and the application of that change to the database at site B).

“Hot-hot” challenges

- How much of your application code will have to be site-aware?
- If you go with “near-synch” for databases, how do you close the “time gap” if a disaster knocks out a site?
 - Asynchronous replication implies that database changes made at one site will be applied to the other site a few seconds later
- Possible to retain even in-flight changes if a site fails?



The tough problems are the most fun to solve!

43

There are a number of challenges associated with the implementation of a “hot-hot” configuration. For one thing, it is highly likely that at least some of your application code will have to be site-aware. In other words, some of your programs are probably going to have to “know” that they are operating in a two-site (for example) configuration, that they are running at present at site A (or site B), and that there are things that they can do at site A that they can’t do at site B (and vice versa).

Another challenge is the “time gap” introduced by “near-synch” bidirectional replication (described on the previous slide). If site A is lost, there will be small amount (perhaps a very few seconds) of data change activity committed on site A and not applied at site B. How can this time gap be closed?

As long as you are at it, you might see if you can devise a set-up that would preserve even data changes made by in flight transactions running at site A if that site were to be incapacitated. That is a tough one, but the toughest problems are often the most fun to solve.

IDUG® 2006 - Europe

Session G07

Get Up: DB2 High Availability & Disaster Recovery

Robert Catterall

CheckFree

rcatterall@checkfree.com

GoFurther

44



Thank you, and enjoy the rest of the conference.