IBM® DB2® Universal Database™ Version 8 and VERITAS® Database Edition / HA for DB2

Accelerating Failover Times in IBM DB2 UDB database environments

February 6, 2004

IBM Toronto Laboratory
VERITAS Software Corporation

ABOUT THE AUTHORS

■ IBM Toronto Laboratory

Nuzio Ruffolo DB2 UDB System Test,
Toronto Laboratory, IBM Canada

Nuzio Ruffolo has been with IBM for over 9 years as part of the DB2 Universal Database System Test Organization (DB2 UDB SVT). Nuzio is responsible for managing a complex environment that includes Linux, UNIX and Windows® workstations along with zSeries® and iSeries™ servers. DB2 UDB testing is performed in this environment based on expected production environment requirements – millions of transactions and thousands of operational tasks repeatedly executed during test cycle. On this team, Nuzio has held various team lead and management positions. Nuzio represents the entire DB2 UDB test organization on the IBM test council and has presented various testing related topics at the annual Centre for Advanced Studies Conference (http://cas.ibm.com). Recently Nuzio has been responsible for testing DB2 UDB in high-availability environments.

■ VERITAS Software Corporation

Ulrich Maurus

As staff engineer for VERITAS UNIX® Edition, Ulrich Maurus oversees DB2 UDB engineering located at the IBM Toronto Laboratory. He has been with VERITAS for 9 years. Before assuming his latest role in engineering, he was senior product specialist for clustering and replication, product manager for SAP Edition, engineer for VERITAS Cluster Server development, and senior consultant for OpenVision and VERITAS Germany.

OVERVIEW

In this paper, we describe the design, implementation and tuning of highly available IBM DB2® Universal Database™ (DB2 UDB) Version 8.1.2 (FixPak 2) environments with VERITAS Database Edition / HA for DB2 as well as VERITAS SANPoint Foundation Suite (VERITAS Cluster Volume Manager and VERITAS Cluster File System) on the Sun Solaris 9 platform. We provide guidelines and recommendations to increase DB2 UDB availability and decrease failover times using DB2 UDB V8.1.2 and the VERITAS solutions mentioned above. Practical considerations regarding design, implementation, testing and performance work with VERITAS Cluster Server, VERITAS Cluster Volume Manager, and VERITAS Cluster File System Release 3.5 MP1 are discussed in the second part of the document. The configurations described in this document are customer deployable and fully supported by IBM and VERITAS. Configurations are also supported on Solaris 7 and 8.

Key Highlights from Failover Tests:

1st Series:

- The first series of tests focused on minimizing the standard DB2 UDB database recovery time and optimizing VERITAS Database Edition / HA for DB2 and associated VERITAS Cluster Server agent for DB2 UDB detection of a system fault.
- **Our tuning modifications of the default DB2 UDB and VERITAS Cluster Server parameters resulted in approximately 35% faster failover time (reduced from 61 to 40 seconds) for a database on two file systems, without compromising the database performance.**

2nd Series:

- In the second series of tests, we examined the impact of various storage configurations on the failover time following a system failure.
- **Using VERITAS Database Edition / HA for DB2 in conjunction with VERITAS Cluster Volume Manager and VERITAS Cluster File System (CVM/CFS), we were able to reduce failover time by up to 86%, thereby providing DB2 UDB recovery that was more than 7 times faster than the solution without CVM / CFS.**
- Failover times remained constant using the components itemized above, regardless of the number of file systems (ranging from 2 to 86), DB2 UDB data containers or control files used in the tests.
- These test results clearly indicate that VERITAS Database Edition / HA for DB2 with VERITAS Cluster Volume Manager/Cluster File System provides fast, consistent failover capabilities for even very large and complicated storage configurations.

3rd Series:

- For the final series of tests, we leveraged the hot standby feature of DB2 UDB by starting the instance database manager on the standby host before the failover occurred.
- Using two modified Cluster Server Agents, we reduced the failover time for all the various storage configurations by approximately 10 seconds. In the clustered storage primary pre-set case, the total failover time was improved by almost 40%.
- **The fastest DB2 UDB failover time achieved was 19 seconds. As with our 2nd series of tests mentioned above, this fast failover capability could be consistently reproduced for a variety of clustered storage configurations, independent of the number of file systems and container files.**

All failover tests were executed while the active server was under a heavy, external application load. The same database file sizes were used for all test runs, thereby ensuring that the failover times were not skewed by inconsistent database sizes. The database server performance was monitored before and after each failover test to determine that there was no significant impact caused by the different storage setups – i.e. clustered versus single server access.

1. PREREQUISITES

We assume that the reader of this paper already has some knowledge of the following topics:

- Business drivers and incentives for implementing highly available computer system environments, and the associated terminology
- Installation and configuration of DB2 Universal Database
- VERITAS Volume Manager, and VERITAS File System
- VERITAS Cluster Server and associated network protocols

2. OVERVIEW OF THE TESTED SOFTWARE COMPONENTS

This section provides a short introduction to the DB2 UDB design as well as the VERITAS software components. For a more in-depth explanation of the software components, refer to the documentation sets delivered with the products or go to: http://support.veritas.com/.

2.1 DB2 UNIVERSAL DATABASE V8.1 FOR LINUX, UNIX AND WINDOWS

IBM DB2 UDB V8 marks the next stage in the evolution of relational databases by providing enhancements in the areas of performance, availability, scalability and manageability. This makes DB2 UDB V8 the database of choice for business-critical solutions such as e-business, business intelligence, enterprise content management, enterprise resource planning and customer relationship management.

In today's global business environment, the ability to access your data at any time is critical. The built-in planned and unplanned availability capabilities of DB2 UDB ensure your business applications are available whenever you need them. Whether switching to a standby database server if an unexpected database failure occurs, or carrying out online maintenance – such as the ability to perform an online reorg - DB2 UDB makes sure all of your business applications remain available. Online utilities such as index rebuild, index create, and table load, as well as configuration parameters that can be changed without stopping the database, mean improved performance and high availability.

For more information about, and a complete list of, the features included in DB2 UDB V8, refer to the documentation available at: http://www.ibm.com/software/data/pubs/.

2.2 VERITAS DATABASE EDITION / HA FOR DB2

VERITAS Database Edition / HA for DB2 is an integrated suite of data, storage and system management technologies that optimize performance, availability, and manageability of DB2 UDB server databases. Besides DB2 UDB- specific utilities for simplified database administration, VERITAS Database Edition / HA for DB2 leverages industry-leading core VERITAS technologies including Volume Manager, File System (with Quick I/O as well as Cached QIO), and Cluster Server.

This paper covers parts of the technologies mentioned above that are relevant to the failover solutions discussed. Additional details regarding these technologies are available from the VERITAS Web site (www.veritas.com) or discussed in the associated VERITAS documentation available as part of the software distribution.

2.2.1 VERITAS Volume Manager (VM) and VERITAS Cluster Volume Manager (CVM)

VERITAS Volume Manager (VM) is a flexible storage virtualization technology that supports Solaris, AIX®, HP-UX, Linux and Windows. VM/CVM allows users to manage physical disks as logical devices called volumes. A volume is a logical device that appears to data management systems as a physical disk partition device. By allowing volumes to span multiple disks, VM enables the management of virtual storage pools rather than actual physical disks. VM allows users to overcome physical restrictions imposed by hardware disk devices by providing an abstraction layer that virtualizes storage and makes managing that storage easier.

VM utilizes software RAID to improve I/O performance as well as protect against disk and other hardware failures. Additionally, VM provides features that enable fault tolerance and fast recovery from disk failure. VM also provides easy-to-use online disk storage management for tasks such as dynamically configuring disk storage while the system is active, ensuring that data remains available to users.

Using logical volume paths (for example /dev/vx/rdsk/db2dg/datavol), VM hides the hardware / operating system / server-specific device information (for example /dev/rdsk/c2t3d12s0 in Solaris notation) from the application, such as a DB2 UDB database. The practical benefits of using logical volume paths become evident in the event of a failover or replacement of the server, because modifications to the database configuration information (e.g., redefining pathnames of database containers) are not necessary. This applies to all DB2 UDB storage management models including database-managed storage (DMS) device / file or system-managed storage (SMS).

VM also supports high availability storage features such as software RAID and multiple, dynamic paths to disk storage. However, our tests only utilized the transparent failover feature of VM, i.e., the failover of logical volumes, which are the logical objects where DB2 UDB containers reside. By focusing on the failover capabilities of VM, we kept our test matrix simple.

The standard VM functionality can be easily extended to enable the VERITAS Cluster Volume Manager (CVM) functionality. This is done by adding a VERITAS license key on each server. CVM allows the concurrent import of disk groups on multiple hosts (currently up to 16). CVM employs a master/slave mode where meta-operations such as volume creations can happen only on the CVM master node. Metadata such as volume definition is distributed over a network link among all cluster members. However, all data traffic from the applications to storage will use local I/O channels.

2.2.2 VERITAS File System and VERITAS Cluster File System

VERITAS File System is a powerful, quick-recovery, journaling file system that provides the high performance and easy management required by mission-critical applications. It delivers scalable performance and provides continuous availability, increased I/O, and up-to-date structural integrity.

A hardware failure that occurs in the middle of a pending I/O activity may leave the file system structure or metadata in an inconsistent state. A complete integrity check using the full "fsck" utility can take minutes or even hours, depending on the size and amount of data that needs to be recovered. This process will cause a significant delay in the application failover time. Because of its design, VERITAS File System's "fsck" utility only involves a log replay after a system failure, which usually only takes a few seconds instead of minutes or even hours. For server configurations using file systems that are tens of Gigabytes in size or greater, a journaling file system such as VERITAS File System is a mandatory item for quick recovery either during reboot or failover caused by a system crash.

For performance reasons, DBAs sometimes choose to implement the database on character/raw devices - known in DB2 UDB terms as DMS (database-managed storage) devices. VERITAS File System's Quick I/O feature provides performance virtually equal to raw devices, but still maintains File System manageability benefits. DBAs can use Quick I/O files as DMS devices to get the best of both worlds. In addition the Cached Quick I/O feature uses the server's extra physical memory as a second-level cache for caching DB2 UDB data blocks. In a 32-bit DB2 UDB environment, this feature can improve DB2 UDB performance significantly beyond that of raw device performance (see other publications for details at http://www.veritas.com/van/products/editiondb2.html).

As with VM, cluster functionality for VERITAS File System (i.e., VERITAS Cluster File System) can be enabled by supplying an additional license key. Using the Cluster File System (CFS) functionality, a file system can be mounted in read/write mode on multiple hosts simultaneously. While the complete I/O data stream uses the local disk controller path, any metadata (file system structure) updates are distributed using the same private network links and communications layer as CVM and VERITAS Cluster Server.

By using the cluster communication links only for metadata information, the traffic across those links is very low compared to other implementations where data is sent over cluster interconnects as well. Moreover, by using the local disk controller for data I/O to files, the latency of the cluster network is eliminated. Especially in DB2 UDB database environments, changes of the file system layout are seldom after the initial implementation. On the other hand, the overall server performance is tightly connected to the latency of the storage subsystem. For this type of application, CFS is superior to network-based shared file system designs such as NFS.

For standard UNIX applications, CFS provides a distributed lock manager (DLM) compatible with the local file system calls. Because all databases implement their own locking mechanism to manage concurrent access to database files, the CFS locking mechanism is unnecessary when used with DB2 UDB. VERITAS Quick I/O bypasses locking at the file level and lets the database server control data access, resulting in enhanced performance, similar to the standard, single-host access test runs.

Note that DB2 UDB currently does not support any type of concurrent database access mode on shared storage but rather an access model called “shared nothing”. Each instance and database partition assumes that it has exclusive access to the file system. The primary reason to use CFS and CVM was to reduce the failover time by eliminating some otherwise necessary steps such as importing disk groups and mounting file systems on a failover target host. The CFS / CVM ability to access disk devices concurrently on multiple hosts, either as a file or as a logical volume, was never used as part of our tests. I/O requests were only performed by a single server at any point in time.

As with CVM, CFS uses a single primary/multiple secondary design. Metadata changes to the file system are routed to the primary node automatically by CFS, while regular I/O operations are handled by individual servers via the locally attached disk channels. In case the primary server leaves the cluster, both CVM and CFS will automatically promote one of the secondary servers to become the primary. Membership and metadata changes are propagated to the primary through LLT and GAB.

While the CVM failover is typically very fast (less than a second in our test environment), the CFS reconfiguration occupies a significant portion of the overall failover time, if more than 10 clustered file systems are used. Because VERITAS File System is journaled by default, a full “fsck” is not necessary. Only a file system log replay is needed on the new primary to bring the file system back online. The log replay is required in order to apply the in-memory image of the file system structure on the new primary server.

The File System consistency check time can be eliminated by designating the DB2 UDB database server as the CFS secondary node, and the failover target system as the CFS primary node. This can be done by executing the `"fsclustadm setprimary <mount_point>"` command on the standby server. In the event of a fault on the standby server, the database server will take over the CFS primary role. If the DB2 UDB database server crashes, the failover time becomes nearly independent of the amount of storage and the number of file systems used by the database, since the new active server is already the CFS primary server. Therefore, the clustered file systems are already mounted and in a consistent state.

Database performance does not depend on which particular node the database server resides. This is because all database I/O requests are performed through local I/O channels. CFS is only exchanging metadata across network links. Therefore, changes to the file system structure, such as allocating space for new container files, will be slightly slower in this configuration.

Moving the CFS primary node inside a cluster does not impact the overall performance of other applications executed on the same servers because the assignment can be done separately for each file system. The distribution and migration of CFS primary or secondary nodes can also be automated using the VERITAS Cluster Server trigger scripts.

2.2.3 VERITAS Cluster Server

VERITAS Cluster Server (VCS) is included with VERITAS Database Edition / HA for DB2, along with the storage foundation technologies such as VM and File System mentioned earlier. Cluster Server is a platform-independent, availability management solution focused on proactive management of service groups (application services). It is equally applicable in simple shared disk, shared nothing or SAN configurations of up to 32 nodes, and compatible with single node, parallel and distributed applications. Cascading and multi-directional application failover is supported, and application services can also be manually migrated to alternate nodes for maintenance purposes. Cluster Server provides a comprehensive availability management solution designed to minimize both planned and unplanned downtime.

Designed with a modular and extensible architecture to make it easy to install, configure and modify, VCS can be used to enhance the availability of any application service with its fully automated, application-level fault detection, isolation and recovery. All ***fault monitors***, implemented in software, are themselves monitored and can be automatically restarted if the fault monitoring processes fail. Monitored ***service groups*** and ***resources*** can either be restarted locally or migrated to another node and restarted. A ***service group*** may include an unlimited number of ***resources***. Various off-the-shelf ***agents*** are available from VERITAS to monitor specific applications such as file services, DB2 UDB and enterprise resource planning.

To make a DB2 UDB database highly available, each system service has a corresponding ***resource***. The system service can be importing a logical disk group, mounting a file system, or starting DB2 UDB itself. On the system level, each type of ***resource*** is associated with a process called an ***agent***. An ***agent*** has implemented various functions called entry points. All agents must have the following mandatory entry points implemented:

- ***online*** - enable / start the resource
- ***offline*** - disable / stop the resource
- ***monitor*** - examine the state of the resource (either online or offline)
- ***clean*** - handle exceptions of the other entry points such as "online ineffective"

Any resource can be in one of three different states, determined by a monitor entry point:

- ***online*** - the system entity associated with the resource is available
- ***offline*** - the system entity is unavailable
- ***unknown*** - the monitor utility could not determine the state, typically caused by a configuration or program error

VERITAS Cluster Server ***resources*** have to be members of another configuration object called ***service group***. The service group attribute ***SystemList*** determines on which computer system of the cluster the associated ***resources*** will be monitored and can be started or stopped. The ***service group*** summarizes the state of all ***resources*** belonging to the group.

Besides online and offline, a ***service group*** can be in partial (for example, "partial online") or transitional states (for example, "waiting to become online"). During normal cluster operation, complete ***service groups***, not individual ***resources***, are activated or stopped. In VERITAS Cluster Server terminology they are called "online service group" and "offline service group", respectively. By specifying ***resource*** dependencies in the Cluster Server configuration file, the user can control the order of ***online*** the ***resources***. For example, you can specify that databases can only be started after all File Systems where DB2 UDB containers reside have been mounted.

VERITAS Cluster Server supports two types of service groups, ***parallel*** and ***failover***. A failover group can only be online (complete or partial) on a single server inside the cluster at any given point in time. If the application or system resource such as CFS supports concurrent data access, then it will be mapped to a parallel service group.

Using group dependencies such as "online local" parallel and failover service group can be combined. For example, a failover group for the DB2 UDB database manager startup can be activated on any node where a parallel group has already mounted the Cluster File Systems that contains the database files. The result is a much faster failover because a hardware fault on the database server would only require that the DB2 UDB related processes be restarted on the target failover system and the database recovered from the transaction log file. In a complete failover environment where a VM disk group has to be imported and File Systems have to be mounted, the restart of those system resources can take a substantial amount of time, thereby prolonging the overall failover process especially with a large/complex storage configuration. The VCS group dependency feature in conjunction with CFS/CVM solves the above-mentioned problem.

2.2.4 VERITAS Db2udb Resource Type / Agent

For this documentation, we briefly discuss the Db2udb ***resource*** type and ***agent*** delivered by VERITAS, which was used during most of the tests.

The agent is distributed as a separate package, ***VRTSvcsdb***. It installs into the same directory tree as the rest of the VCS distribution:

- configuration samples and type definition files into /etc/VRTSvcs/conf
- binaries and executable into /opt/VRTSvcs/bin/Db2udb

By using a separate type definition file ***Db2udbTypes.cf,*** which can be added to the cluster configuration using an include statement, the DB2 UDB related agent files may be used and upgraded independent of the current VCS version. For example, the agent used for our test (3.5) can be combined with Cluster Server Release 1.2.1 up to 3.5.1.

The complete specifications of the DB2 ***agent*** are documented in the "VERITAS Cluster Server Enterprise Agent™ for DB2 UDB Installation and Configuration Guide". Nevertheless, here is a short introduction to the mandatory entry points of the Db2udbAgent used for our tests:

- ***online***
  Start the DB2 UDB database manager using ***db2start***.

- ***offline***
  Stop the DB2 UDB database manager using ***db2stop force***.

- ***monitor***
  Check for presence of DB2 UDB database manager processes using ***db2nps*** (if installed), or the ***ps | grep*** construct. If enabled by resource attribute ***IndepthMonitor,*** connect to database and query table "sysibm.sysdummy1".

- ***clean***
  Use ***db2nkill*** (or UNIX command "kill -9") to terminate any active DB2 UDB database manager process on the local server.

2.2.5 VERITAS Communication Protocol Layer

Hardware fault detection of a cluster node is not perceived by the VERITAS Cluster Server components described in the sub-chapters ahead, but rather through the unique cluster communication protocol layer.

On all open systems, this communication software is distributed as two separate packages. In Solaris environments, the packages are: ***VRTSllt*** for Low Latency Transport (LLT) and ***VRTSgab*** for Group Membership and Atomic Broadcast (GAB). Both packages are tightly integrated operating system kernel extensions (loadable modules for Solaris) and not available from VERITAS without either Cluster Server and / or SANPoint Foundation Suite (the current VERITAS solution that includes CVM and CFS).

Configuration details of LLT/GAB are covered in the VERITAS Cluster Server documentation set, but since the tuning of this communication layer has significant impact on the failover time, we introduce some basic design features of the cluster communication protocol layer here.

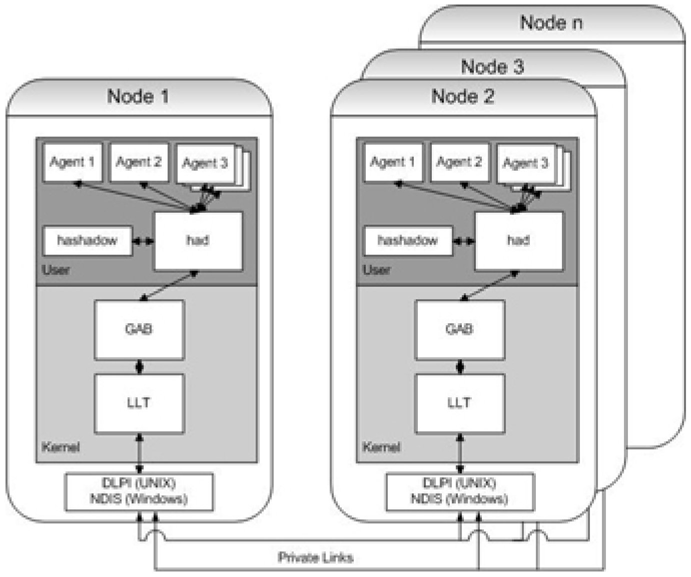


Figure 2.1 VERITAS Cluster Communication Design

Figure 2.1 above shows the flow of information between two cluster nodes. All nodes are attached to at least two private network links. At this time, VERITAS supports only IEEE 802.3 network connections (better known as Ethernet).

The LLT protocol does not encapsulate the information into IP frames like many other applications and even cluster software protocols, but rather accesses the network hardware directly through the Data Link Provider Interface on all UNIX platforms. LLT is a reliable protocol, meaning that all data frames sent by one node over the private links have to be acknowledged by the recipients. In the event of a single network connection failure, incomplete or unacknowledged information will be retransmitted over the remaining links. For high availability implementations, multiple, redundant network connections among all members of a cluster are mandatory.

If no traffic from another node can be observed inside the predefined time interval ("peer inactivity"), the next protocol layer - currently only GAB - will be notified. Any network traffic generated by GAB is used to determine the state of the network connections. If GAB does not generate traffic, the LLT module can create minimum-sized packages on its own and send them to peer systems at intervals that are predefined by the heartbeat timer. The default values for peer inactivity are 1600 time ticks of 1/100ths of a second each. The default heartbeat value is 50 ticks.

While LLT is responsible for the reliable transmission and monitoring of the cluster interconnects, only GAB is able to analyze and act in case of hardware failures. The most important function of the second communication layer (GAB) is to establish cluster-group membership information. Inside the LLT configuration, all servers that form a cluster are assigned the same cluster ID (an integer number between 1 and 255) as well as a unique node ID (an integer number in the range of 0 to 31). When a cluster is formed, the GAB layer on each node will track the currently visible node IDs and form a membership. All member-cluster nodes will share all membership information, and any new server joining an existing cluster will need to register and can then access the most recent copy of the information pool.

To support multiple cluster applications, GAB utilizes a port design similar to the IP protocol. Currently up to 24 different ports can be supported, with each port representing a unique application in the same cluster. The ports are identified by lowercase single characters from "a" to "z". As soon as the GAB module is activated, it will try to register the server on port "a" with any other node accessible through the predefined network connection lines.

The following table contains a partial list of all GAB ports that were in use during our tests. VERITAS maintains a list of all already assigned ports, available on request.

a	Generic GAB
f	Cluster File System
h	VERITAS Cluster Server
u v w	Cluster Volume Manager

Table 2.2 Preassigned GAB ports used as part of the tests

Once a cluster service is registered, GAB will then ensure that application data like cluster resource states or file system locks are replicated to all members of the cluster. This is implemented by using atomic broadcast messages. "Atomic" is used in the sense that the related information is either received and acknowledged by all members of a cluster or discarded by all members.

For our test scenarios, the GAB "stable-timeout" parameter was an important factor in determining the overall failover time. As soon as LLT detects that the communication lines to a peer server are all inactive, it will inform GAB of the membership change. After the stable-timeout period is exhausted, GAB will propagate this information to all members. The cluster application, such as CFS or VCS, will then determine how to handle a membership change.

3 FAILOVER TESTS

The tests were executed on two clustered Sun E450 servers, each configured with four UltraSPARC II processors and one Gigabyte of memory. As indicated in Figure 3.1 below, two private network links were implemented using crossover Ethernet cables connected to separate 100 BaseT ports.

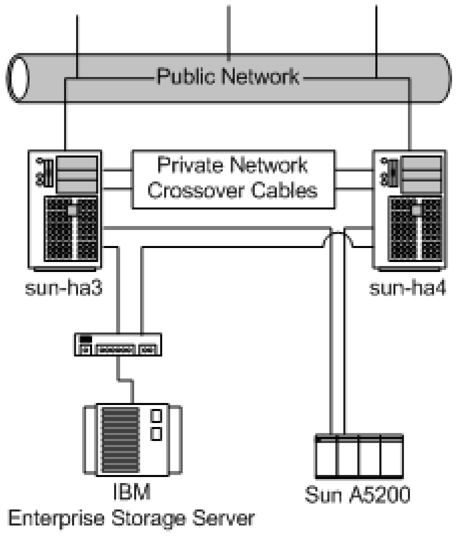

Public Network
Private Network Crossover Cables
sun-ha3
sun-ha4
IBM Enterprise Storage Server
Sun A5200

Figure 3.1 Hardware Setup

3.1 STORAGE AND DATABASE CONFIGURATION

To determine the impact of the storage configuration on the DB2 UDB failover time, three different disk layouts were used:

1. IBM Enterprise Storage Server (ESS, also known as Shark); a single RAID-5 LUN of 110 Gigabytes; 2 VM Volumes with 2 VERITAS File Systems; Quick I/O enabled

2. IBM ESS with 100 x RAID-5 LUNs of 853 Megabytes each; 86 VM Volumes with 86 VERITAS File Systems; Quick I/O enabled

3. Sun A5200 with 22 x 36.4 Gigabytes disk drives; 14 VM Volumes with 14 VERITAS File Systems; Quick I/O enabled

The database instance home files were put on a separate file system/volume of 10 Gigabytes, mounted under /db03/svtdbm. The DB2 UDB binaries were installed on the local system disk of each server. Depending on the storage layout, the database container files were located on a single large file system or stored in multiple file systems. Because of a limitation of VERITAS File System in clustered mode, the file systems for the DB2 UDB container files were all mounted with separate sub-directories under /db03. We used the same mount hierarchy for all configurations for a more accurate comparability of the test results.

To determine the impact of VM configuration sizes on failover times, the 22 x 36.4 Gigabytes disks of the A5200, used in layout No 3 above, were subdivided into 137 subdisks of 256 Megabytes each. Then for every database container and control the file directory, separate volumes (a total of 13) were created using software RAID-0+1 for performance and redundancy. The idea was to inflate the number of VM objects (for our test the disk group contained 3,104 objects), simulating a far larger DB2 UDB storage layout than the one used for these tests. A VERITAS file system was created on each volume.

Using this storage configuration and associated mount points, we could easily switch between failover and parallel setup. Because VERITAS CVM and CFS use exactly the same data and disk layout, only the VERITAS Cluster Server configuration had to be modified between the test runs. The complete DB2 UDB database installation and configuration as well as storage layout could be reused, thereby making the relative performance results of clustered and failover tests comparable.

All DB2 UDB database files as well as control files resided on VERITAS File Systems. For the database container files the "qiomkfile" utility was used to allocate the disk space. The qio files were then introduced to DB2 UDB as database managed storage (DMS) of type "device" using the pathname to the logical links created by "qiomkfile" (refer to the VERITAS "Database Edition for DB2 - Database Administrator's Guide" for more details regarding Quick I/O available at http://support.veritas.com/).

The hardware was not altered during the complete test sequence. For example, the IBM ESS was still attached (but not used) while tests were running with the Sun A5200 as a storage device. Only the resources for the logical disk management and mounts were disabled or removed from the Cluster Server configuration. For the CVM and CFS tests, exactly the same network and disk configuration were used.

A database layout was created to support an OLTP-like workload that is commonly used in the DB2 UDB System Verification Test cycle.

Four buffer pools of 16 Megabytes each were created, for a total database buffer pool space of 64 Megabytes. Page sizes of 4, 8, 16 and 32 KB were used, with one unique page size for each of the four buffer pools.

The database layout consisted of the following DMS tablespaces using device type containers of various page sizes.

- 4 temporary table spaces of 4KB, 8KB, 16KB and 32KB page sizes totaling 20 GB
- 4 regular table spaces of 4KB, 8KB, 16KB and 32KB page sizes to store data totaling 20 GB
- 4 regular table spaces of 4KB, 8KB, 16KB and 32KB page sizes to store index data totaling 8 GB
- 1 long/large table space of 4KB page size to store long data columns totaling 40 GB

Other database objects (TABLES, VIEWS and INDEXES) were then created across these table spaces. The database is made up of 1,286 TABLES (each of which has a corresponding ALIAS), 24 VIEWS and 51 INDEXES.

The width of the tables used in this workload ranged from a single column to 203 columns. The average column count for all tables was 10.

Once the objects were created, a random data generator was used to populate the objects with data. An initial database size of approximately 15 GB was used during the testing.

The database load was created by an external AIX system accessing the database using a virtual IP, which enabled transparent reconnection to the applications in case of a failover. The DB2 UDB clients from the AIX system were Java^{TM} applications that accessed the DB2 UDB server on Solaris via the JDBC interface.

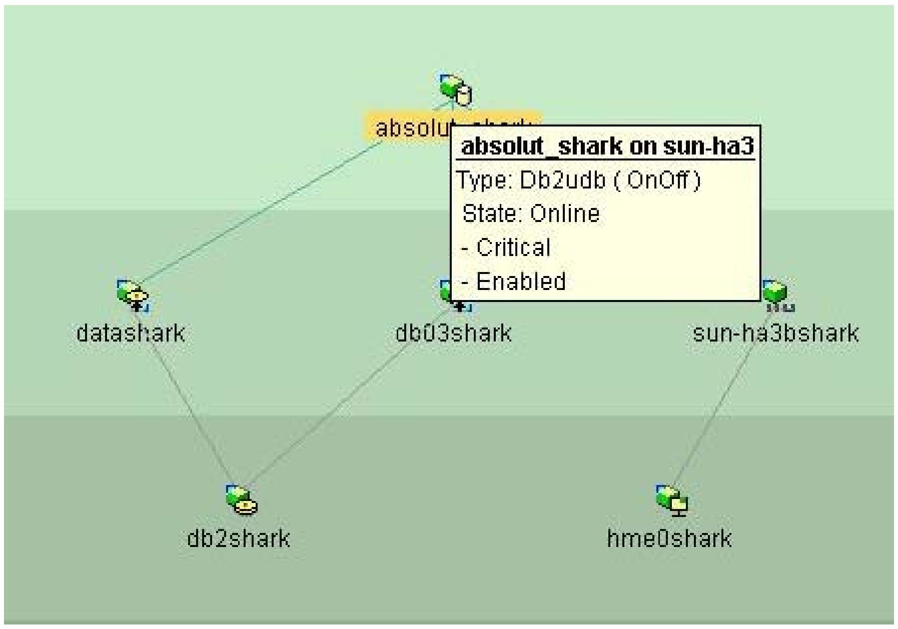
Below is a breakdown of the types of SQL commands included in the client workload:

SQL Type	**Raw Count**	**Percentage**
SELECT	1664	41.8
SEARCHED UPDATE	1492	37.5
SEARCHED DELETE	381	9.6
LOCK TABLE	120	3.0
INSERT	324	8.1
	3981	**100 %**

Table 3.2 DB2 UDB workload mix

The Java application from the client would randomly select an SQL statement out of this pool to execute against the server and would execute concurrent threads to simulate a multi-client load against the server. To better mimic an OLTP workload, two invocations of the Java application were executed. The first invocation would only run the SELECT statements and would make up 80% of the overall client count. The second invocation would run the rest of the SQL commands in the pool and make up the remaining 20% of the client count. The workload breakdown would then be 80% read and 20% INSERT/UPDATE/DELETE. On average over 100,000 rows were accessed per minute by the clients.

3.2 BASELINE CLUSTER CONFIGURATION FOR DISK GROUP FAILOVER



absolut_shark on sun-ha3
Type: Db2udb (OnOff)
State: Online
- Critical
- Enabled

Figure 3.3 Screen shot of VERITAS Cluster Server Failover Service Group Configuration

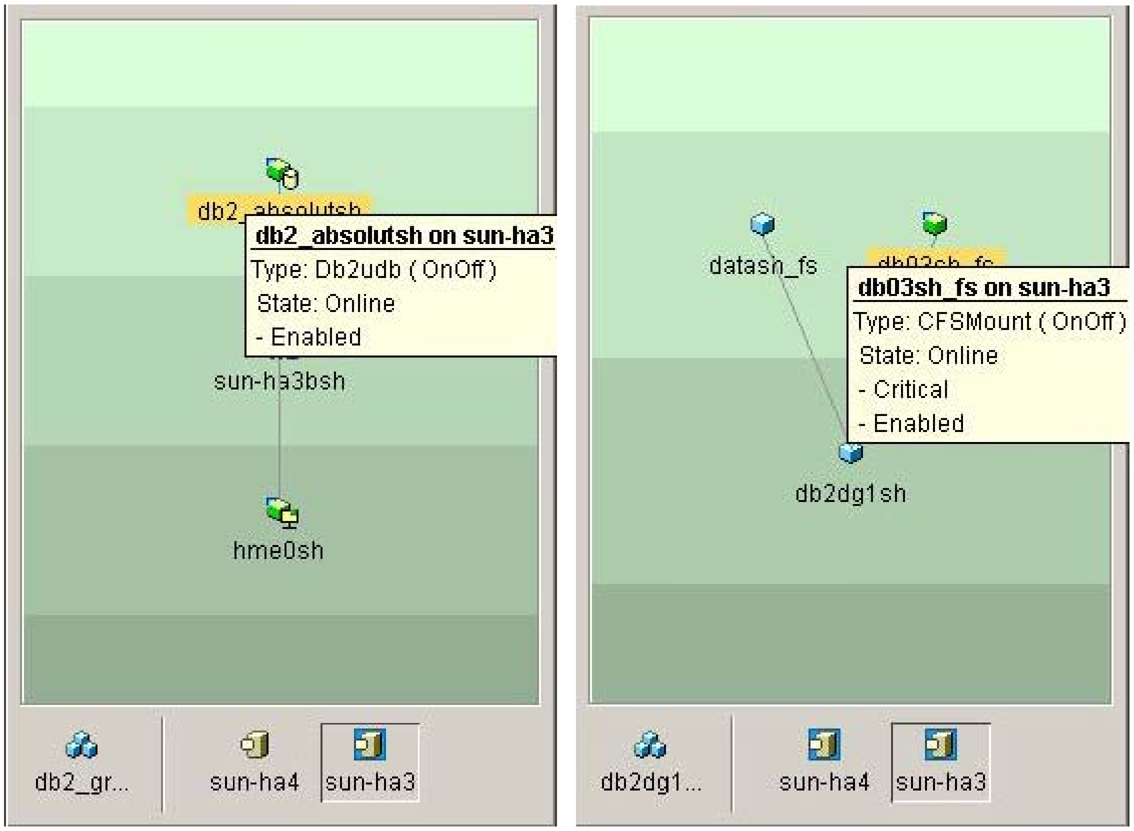
For the first test run, we used a standard VM setup where the disk group is imported only on a single node. Figure 3.3 shows the cluster server group layout, typical for DB2 UDB failover configurations. The ***DiskGroup*** resource (db2shark) has dependencies to resources of type ***Mount*** (datashark for the DB2 UDB container files, db03shark for the control file directory). The resource called ***absolut_shark*** would start the database only if all linked resources were in a state online. Besides the ***Mount*** resources, this is another resource of type ***IP*** named sun-ha3bshark initializing a virtual IP interface. DB2 UDB clients using the virtual IP address to connect to the database server do not need any network reconfiguration in case the complete resource group is switched between different cluster nodes. The hme0shark resource is of type ***NIC*** and represents the physical network interface card on each server, which is used to create the virtual IP address. The complete group represents all system services necessary to access the DB2 UDB database used for our test runs.

While Figure 3.3 shows only the two-file system setup, for the 14 and 86 configurations an equivalent number of ***Mount*** resources were added and linked to the ***DiskGroup*** and ***Db2udb*** resources. The configuration details can be found in Appendix D: VERITAS Cluster Server Configuration Files.

In the event of a system crash, the takeover node has to clear the existing ownership information associated with the crashed node in the disk group header. Then the disk group has to be imported. This process includes the creation of two special devices (raw and blocked) for each volume inside the /dev/vx directory tree.

The VERITAS Cluster Server ***MountAgent*** responsible for the file system recovery can perform the file system repair and mount for multiple storage devices in parallel. There is, however, a limit on the number of threads for a single process such as the ***MountAgent*** that is imposed by the operating system. In our test case we observed no more than 10 parallel online scripts being executed.

3.3 SHARED STORAGE CONFIGURATION USING VERITAS CLUSTER VOLUME MANAGER AND VERITAS CLUSTER FILE SYSTEM



db2_absolutsh on sun-ha3
Type: Db2udb (OnOff)
State: Online
- Enabled

db03sh_fs on sun-ha3
Type: CFSMount (OnOff)
State: Online
- Critical
- Enabled

Figure 3.4 Screen shot for VERITAS Cluster Server, Cluster File System, Service Group Configuration

Even though DB2 UDB is designed as a shared-nothing storage access application, we ran tests using the VERITAS cluster extensions for Volume Manager (CVM) and File System (CFS). CVM and CFS significantly improves DB2 UDB failover time in the event of a crash. Because the disk group is concurrently imported on all cluster nodes and the file systems are already mounted in place, only the application and the virtual Internet address have to be restarted on the failover target server.

The VERITAS Cluster Server configuration contains three separate ***service groups,***: two ***parallel*** and one ***failover***. Figure 3.4 shows the DB2 UDB database related service groups db2_group1sh and db2dg1_shared. The failover group db2_group1sh enables the DB2 UDB database manager including the virtual IP address and was active on server sun-ha3 when the screen shot was taken. Import of the disk group and mounting of file systems for the database is done by the resources that are part of the parallel group db2dg1_shared. Because those services are clustered volume manager and clustered file systems, they can be active at any time on both nodes. The blue icon background of sun-ha3 and sun-ha4 displays that the group is online on all servers.

CVM and CFS need some infrastructure processes running in the background on each node. These processes are located in the group named CVM, which is automatically created by the VERITAS SANPoint Foundation Suite/HA installation (for details on this group, refer to VERITAS SANPoint Foundation Suite 3.5 Installation and Configuration Guide available at http://support.veritas.com/).

Using a separate parallel group for managing the clustered disk group and file systems allows more flexible management, extension, and reconfiguration of the cluster. Such a design enables multiple clustered disk groups to be imported and exported on each node independently.

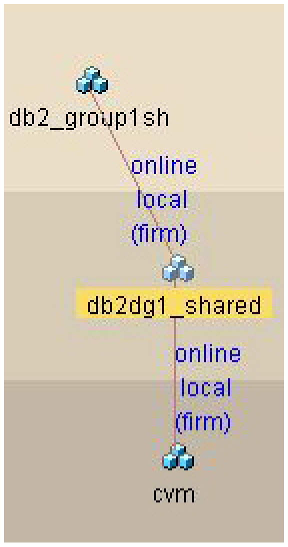


Figure 3.5 Cluster Service Group Dependencies

Using a group dependency ***Online Local Firm,*** as shown on Figure 3.5, the disk group will only be imported and the file systems mounted if the CVM / CFS daemon processes are present on the node. While the file systems containing the database containers and control files are mounted on both nodes, the DB2 UDB database can only be active on a single node at any given time. Using a failover group with ***Online Local Firm*** dependency defined in the CVM / CFS service group guarantees that DB2 UDB is only started if the necessary disk group and file system resources are available on that node. The failover group attribute (it's actually ***Parallel = False***) prevents a start of the resources on more than one server.

In case a system fault occurs, VERITAS Cluster Server will just take care of starting the failover group on the target node. The CVM / CFS group should already be online and Cluster Server will detect no change in state for the ***CVMDiskgroup*** and ***CFSMount*** resources. The necessary reconfiguration of the CVM / CFS assignments will be triggered by the GAB group membership change outside the control of VERITAS Cluster Server.

3.4 TUNING LLT AND GAB TIMER

The communication protocol timers add a fixed period to the complete failover time. While the detection of a faulted cluster node is detected inside a LLT heartbeat interval - by default 50 msec - this information is not propagated to the cluster server or CVM/CFS before two other timer values are exhausted. After the installation of the VRTSgab and VRTSllt packages, the default values for these timers are set to 16 seconds and 5 seconds, respectively.

To decrease the overall failover time, we changed both values as follows:

- LLT peerinact = 3 seconds
- GAB stabletimeout = 2 seconds

The reason for the relatively long default values is to avoid failover attempts caused by an overloaded server. Especially for single processor machines, there is a slight chance that the LLT module is not executed inside a heartbeat interval. Even though the complete communication layer is implemented as kernel modules (on all commercial UNIX operating systems such as Solaris, AIX and HP-UX) which have scheduling precedence to user process space application such as DB2 UDB, there is still a chance that the LLT driver is not executed on time. One possible cause can be another driver such as disk I/O, which may be in a non-interruptible sequence and does not complete because of a disk controller error, thus blocking a CPU. The very conservative LLT and GAB timer address this problem by canceling a failover sequence in case a node recovers before both timers are exhausted.

In our test environment we generated heavy system loads through the external DB2 UDB database applications, which resulted in response times of 20 to 30 seconds for interactive shell command execution. Still we never observed any missed heartbeat notification by LLT or unrelated membership reconfiguration by GAB. One reason was the use of four-processor systems in our test configurations, which guarantees a better scheduling preference for Kernel modules such as LLT and GAB compared to single processor systems. This is especially true in case the system is running a high user application load. On one occasion a complete loss of the VERITAS Cluster Server agent framework was experienced, resulting in an automatic disabling of all service groups on the other node. But as long as GAB does not signal a membership change, VERITAS Cluster Server will not online any failover group on another node. This prevents a “split brain” problem, where a two-cluster member may independently run the same application, or, as in our test environment, the same DB2 UDB instance.

As a precaution we used the non-default gabconfig parameter -j, which will force a node to panic in case it rejoins an existing cluster of which it was a former member. Our test showed that LLT and GAB protocol worked reliably at all times and across all of the various system loads. **Using the tuned timer we were able to reduce the protocol-related failover detection from approximately 21 seconds to 5 seconds.**

3.5 TUNING THE CLUSTER SERVER DB2UDBAGENT

For the Failover Tests we made a small modification to the "online" entry point script of the Db2udbAgent. By default, the Agent does only an instance start using the "db2start" command. In this case the database is opened by the first application trying to connect to the database or resume operation after a restart of the DB2 UDB server processes (db2sysc). Because we enabled for all tests the In-Depth Monitoring feature of the agent, which does a "db2 connect to <database>" as part of the monitor entry point, we got inconsistent time periods for the overall service group restart in case of a failover.

One reason for this finding seemed to be that the monitor script connection competed with network load application clients reconnecting at the same time. To suppress the deferred opening of the database, we added a "db2 restart database $DBName; db2 terminate" to the same command line that was doing the "db2start" command as part of the agent online-script. This modification did result in more speed and less fluctuation in the complete crash recovery time.

Because we observed another delay between the instance start and the database open using the standard Db2udbAgent, one series of tests was executed using a highly customized implementation (see Appendix A: DB2 UDB Instance Start Before Database). For these tests the ***db2start*** and ***db2 database restart*** were executed by two different agents. This modified VCS code was only verified in our limited test environment with only one single partition database configuration.

We confined our tests to failover time optimization and did not verify the ***DB2udbAgent*** ability to restart an instance on the same node. In case of a local software fault such as db2 server processes dying on their own account, the observed restart may significantly exceed the values we measured for a failover scenario. In case of local restart attempts, the agent monitor interval determines the service group outage. Because ***monitor*** entry point is executed by default only every 60 seconds, the possible reconnect time after a software crash can vary by a complete monitor interval.

3.6 TUNING DB2 UDB FAILOVER TIME

Numerous DB2 UDB parameters can affect overall failover time and can be tuned. The parameters tuned for this test to minimize failover time are described in the sections below.

3.6.1 Tuning DB2 UDB RESTART TIME

The following table outlines the database configuration parameters that were changed to minimize the amount of time it takes to restart the DB2 UDB database on the stand-by node in the cluster after a failover.

Parameter	Default Value	Tuned Value
LOGBUFSZ (4KB)	8	4
CHNGPGS_THRESH (%)	60	20
NUM_IOCLEANERS	1	20
NUM_IOSERVERS	3	10
LOGFILSIZ (4KB)	1000	500
LOGPRIMARY	3	250
LOGSECOND	2	0
SOFTMAX (%)	100	1
INDEXREC	SYSTEM (RESTART)	ACCESS

Table 3.6 Tuned Database Parameter

With the exception of INDEXREC, the parameters in the table above were tuned to have more pages written to disk prior to a failover and to minimize the number of DB2 UDB logs that need to be replayed in the event of a crash recovery. When more pages have been written to disk, recovery of the database on the stand-by system is faster because the database manager can rebuild more of the buffer pool from disk instead of having to replay transactions from the database log files. For this reason, the page cleaner is activated if the size of the log that would need to be read during recovery exceeds the following maximum:

```
   LOGFILSIZ * SOFTMAX
```

The INDEXREC parameter indicates when the database manager will attempt to rebuild invalid indexes. Setting this parameter to ACCESS will rebuild invalid indexes during index access, rather than at db2 restart time.

For more information about these parameters, refer to the following DB2 UDB V8 documentation:

Administration Guide: Performance (ftp://ftp.software.ibm.com/ps/products/db2/info/vr8/pdf/letter/db2d3e80.pdf)
Data Recovery and High Availability Guide and Reference
(ftp://ftp.software.ibm.com/ps/products/db2/info/vr8/pdf/letter/db2hae80.pdf)

3.6.2 Tuning DB2START TIME

In the DIAGPATH (i.e., $HOME/sqllib/db2dump) a file called db2eventlog.<nodenum> is written at db2stop time or when an instance crash occurs. This file contains a recent history of high-level events that occurred within the DB2 UDB engine that can be used to help debug problems. This facility is known as the DB2 UDB Event Log. There is some overhead associated with setting up the event log at db2start time. This impact can be minimized by setting the DB2_EVENT_LOG_CONFIG DB2 UDB registry parameter to either OFF or MINIMUM.

This test was performed with this parameter set to OFF

```
db2set  DB2_EVENT_LOG_CONFIG=OFF.
```

This turns event logging off completely.

On the system used for this test, we observed a savings on db2start time of up to 10 seconds, which decreased our overall failover times (for the tests in which DB2 UDB was not already started on the standby system).

```
[133] [svtdbm@sun-ha3] /export/home/svtdbm > db2set
DB2COMM=tcpip
[134] [svtdbm@sun-ha3] /export/home/svtdbm > time db2start
07-28-2003 10:27:04     0   0   SQL1063N  DB2START processing was successful.
SQL1063N  DB2START processing was successful.

real    0m13.39s
user    0m0.42s
sys     0m0.93s
[135] [svtdbm@sun-ha3] /export/home/svtdbm > db2stop
db2set db207-28-2003 10:27:12     0   0   SQL1064N  DB2STOP processing was successful.
SQL1064N  DB2STOP processing was successful.
[136] [svtdbm@sun-ha3] /export/home/svtdbm > db2set db2_event_log_config=off
[137] [svtdbm@sun-ha3] /export/home/svtdbm > db2set
DB2_EVENT_LOG_CONFIG=OFF
DB2COMM=tcpip
[138] [svtdbm@sun-ha3] /export/home/svtdbm > time db2start
07-28-2003 10:27:32     0   0   SQL1063N  DB2START processing was successful.
SQL1063N  DB2START processing was successful.

real      0m4.67s
user      0m0.48s
sys       0m0.89s
```

Setting this parameter to MINIMUM will start up event logging with the smallest possible event log. Our tests showed that db2start would take up to 0.5 seconds more with DB2_EVENT_LOG_CONFIG set to MINIMUM over the db2start time with DB2_EVENT_LOG_CONFIG set to OFF.

3.6.3 Tuning DATABASE ACTIVATION TIME

By default, an event monitor is created and enabled for each database. This event monitor, named DB2DETAILDEADLOCK, starts when the database is activated and will write to files in the database directory. You can avoid the overhead this event monitor incurs by deactivating it and (optionally) dropping it.

To list the event monitors, execute the following commands:

```
db2 connect to database absolut
db2 select * from syscat.eventmonitors
```

To deactivate and drop the DB2DETAILDEADLOCK event monitor, use this command:

```
db2 set event monitor db2detaildeadlock state 0
db2 drop event monitor db2detaildeadlock
```

On the system used for all our tests, we observed a failover time savings of up to 1 second.

3.7 FAILOVER TIME TEST RESULTS

Using the configuration variations and tuning discussed above, we observed several improvements in the failover times for the three different storage configurations.

3.7.1 Reducing the Failover Time through Tuned Cluster Timer and DB2 UDB Configuration

For our first test series we used a failover storage configuration as per description 3.2 Baseline Cluster Configuration for Disk group Failover. Only the VERITAS Cluster Server, and the DB2 UDB timer and configuration parameter were tuned.

	Membership Change Detection		Storage Recovery		Service Group Restart		DB2 UDB Recovery	
	LLT	GAB	VM	FS	IP	Db2udb	Detect	Complete
Default Parameter	15	19	24	31	21	109	49	61
Tuned Parameter	3	5	10	18	8	52	34	40

Table 3.7 Out of the Box versus Tuned Parameter Configuration for 2 Volume & File Systems Storage

The columns of the table represent the completion of a specific task in the failover process. All values are averaged for 10 test runs per configuration. The standard deviations from the values in the table were less than 10% of the measured value for each step. Here is the column description in more detail:

- **Membership Change Detection**
  The amount of time it takes until the VERITAS communication layer accepts the fault of a cluster member. This time is determined by a LLT and GAB configuration parameter and was very consistent throughout all test runs. These values are extracted from /var/adm/messages on the standby node.

- **Storage Recovery**
  For tests with shared (non-clustered, without CFS/CVM) storage configuration, access to the disks is restored under control of the VERITAS Cluster Server ***DiskGroupAgent*** and ***MountAgent*** processes. This step includes the import of the logical disk group and the mount of all file systems storing the DB2 UDB data container and control files. The values are extracted from the Cluster Server engine_A.log file.
  For tests with clustered storage, the CVM and CFS reconfiguration is triggered by a GAB membership change notification. In this configuration, the associated ***CVMDiskGroup*** and ***CFSMount*** resources should already be onlined prior to the failover. Therefore, these two resources do not require any recovery. The observed completion time is primarily governed by the log replay of the CFS mounted file systems. The values are extracted from /var/adm/messages.

- **Application Restart**
  This two-step process is executed by two VERITAS Cluster Server agents. First the IPAgent creates a virtual address entry to make the failover transparent to the DB2 UDB network client: they can reconnect to the same TCP/IP address and port after the failover. The virtual IP activation is always instantly executed by Cluster Server as soon as the membership change occurs.
  The second step is implemented by the Db2udbAgent, which executes a "db2start" of the instance as part of "online". Because the associated resource will only be marked with the state "online" after the first successful completion of the "monitor" script, the deviation for this step is larger than for any other part of the failover. The explanation for the very long duration of this step is the fact that "monitor" has to do a "db2 connect to <database>" (in-depth monitoring was always enabled), which competes with the reconnecting external IP clients. All values were extracted from the VCS engine_A.log file.

- **DB2 UDB Log entries**
  Because of the large deviations and delayed status change of the Db2udbAgent, we decided to extract the real restart completion of the DB2 UDB database using the db2dump file. To reveal the impact of the DB2 UDB restart time, we include two points in time. The first value "Recovery Detection" represents the point in time when the database engine detects that the database is in an inconsistent state. The second value "Recovery Completion" marks the point in time when the database engine has completed the recovery and will accept connection requests from clients. This value provides a much more accurate estimate of the actual inaccessibility of the database than that derived from the Cluster Server state change of the Db2udb resource. The "Recovery Detection can be minimized by starting the db2sysc processes prior to the failover, as discussed in "Appendix A: DB2 UDB Instance Start Before Database Open".

Note that the unavailability of the DB2 UDB database is not the sum of all intermediate steps necessary to restart the application. Some of the steps, such as assigning a virtual IP to the public network, disk group import and file system mount, are executed by Cluster Server in parallel. The complete resource reconfiguration and application failover is always triggered by the GAB group membership change detection.

In our tests we used a configuration where the virtual IP used by applications on network clients to access the database is activated prior to the database restart. One advantage is faster database access by the clients after failover of the database server is completed. During our tests we checked by using the external client reconnect time as well as an interactive "db2 connect to absolut" command, that a connection to the database was available as soon as the DB2 UDB recovery process had been completed. This earlier point in time can be found in the DB2 UDB log "~db2user/sqllib/db2dump/db2diag.log". The Cluster Server ***Db2udbAgent*** reported the state ***online*** of the database several seconds after the recovery process completion. We identified one reason for the delayed change in state information being a race condition between the ***Db2udbAgent*** attempt to connect to the database as part of the ***monitor*** entry point and the reconnect requests of all workload clients.

3.7.2 Reducing the Failover Time using VERITAS SANPoint Foundation Suite

The following floating bar charts illustrate the impact of the storage setup (clustered or shared) on the overall DB2 UDB inaccessibility for clients. The results were generated using the 14-volume and file system layout with tuned LLT, GAB and DB2 UDB parameters. **As Figure 3.8 shows, the failover time dropped from 77 seconds to 36 seconds, so for our hardware setup we cut the time our DB2 UDB database was inaccessible by more than a half.**

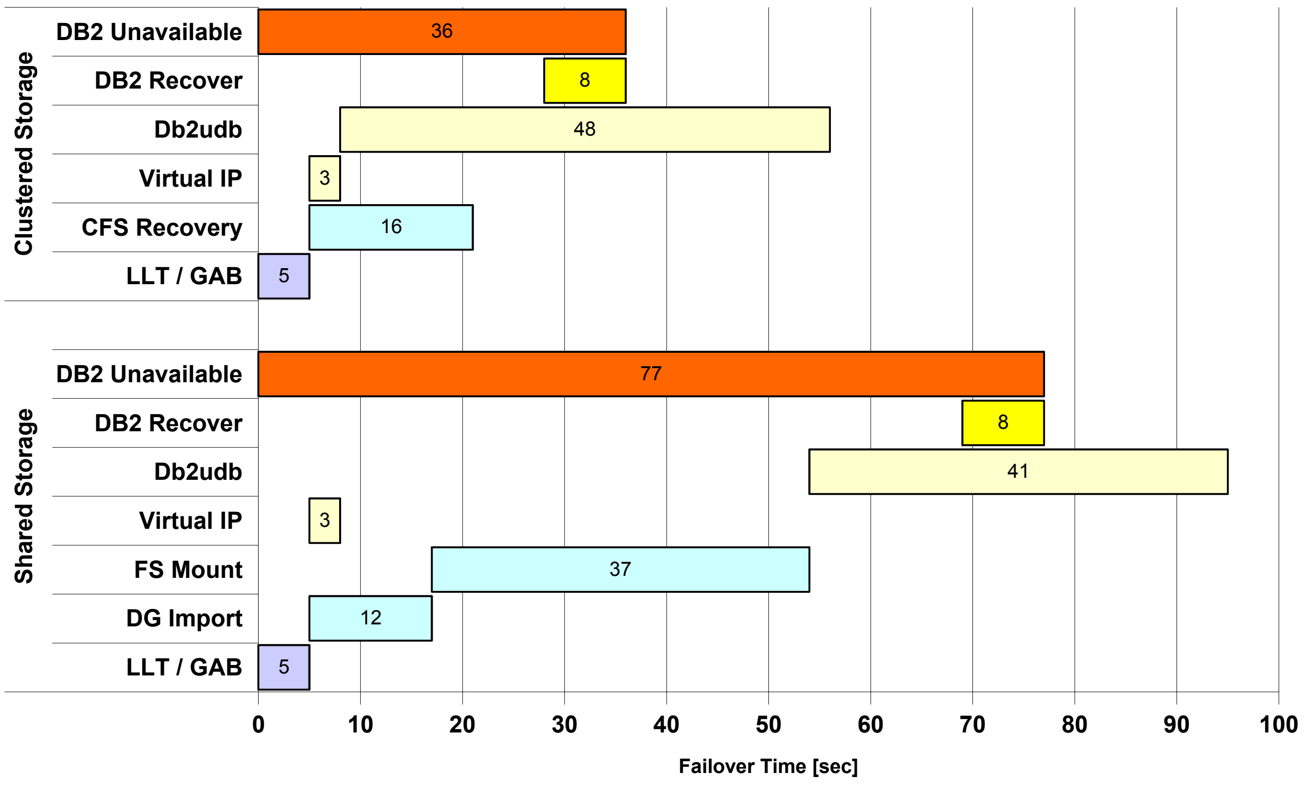


Figure 3.8 Failover Timeline for 14 Volumes / File Systems Configuration

The time frame to import the disk group more than doubled from 5 to 12 seconds between the two-volume setup and the 14-volume test run. (The 5 seconds are the difference between the VM and GAB column in table 3.7 above, the disk group import start with no delay after the GAB group membership change has been propagated.) One reason is the "artificially" increased number of VM objects (see 3.1 Storage and Database Configuration). But the disk group import is still less than 15% of the complete DB2 UDB restart time. If more hardware disk devices as well as logical volumes with more complex structures are used for the database, the disk group import will take significantly longer. For this scenario the CVM reconfiguration time, which is hardly detectable in case of a failover, will represent a larger advantage to the non-clustered solution than in our test cases.

The 14-volume setup shows a large increase in the observed failover time caused by the file system check and mounts to the two-volume setup in the previous test run from 8 to 37 seconds. With 14 volumes in the configuration, we exceeded the default number of threads of the VCS ***MountAgent,*** which is set to 10. We observed that the majority of the file system checks were started in parallel by the agent, but the executions of the monitor scripts of the MountAgent were deferred until sufficient ***online*** scripts had terminated.

The maximum number of threads for a single process is limited by operating system variables, in case of Solaris in the range of 16 to 64. But even for multiprocessor servers like the four-way systems used for our tests, increasing the number of threads may not accelerate the execution time of a task such as mounting 14 file systems, because all mount and file system check commands are using exactly the same hardware resources (disk I/O controller and fibre channel) as in our configuration. So we decided to use the default values provided by VERITAS Cluster Server.

3.7.3 Presetting the Standby Server as Cluster File System Primary

When growing the number of file systems used for database containers, the clustered storage takes less time to reconfigure compared to standard, shared storage setups. During our tests we observed that CFS used about one second to verify the consistency of every clustered mounted file system during a failover. This happened only if a CFS secondary had to be promoted to become a primary server.

If the failover target host was already the CFS primary server, we observed no time delay and the DB2 UDB database restart could be executed right after the cluster membership change was completed

To verify our observation we increased the number of file systems to 86, still using the same ABSOLUT database layout and workload. To force the standby server to become the CFS primary, we executed the following command:

```
# fsclustadm setprimary <mount_point_directory>
```

CFS allows the primary and secondary roles to be assigned to each mount point individually. For our single database configuration, the standby server was assigned the primary role for all container and control file systems.

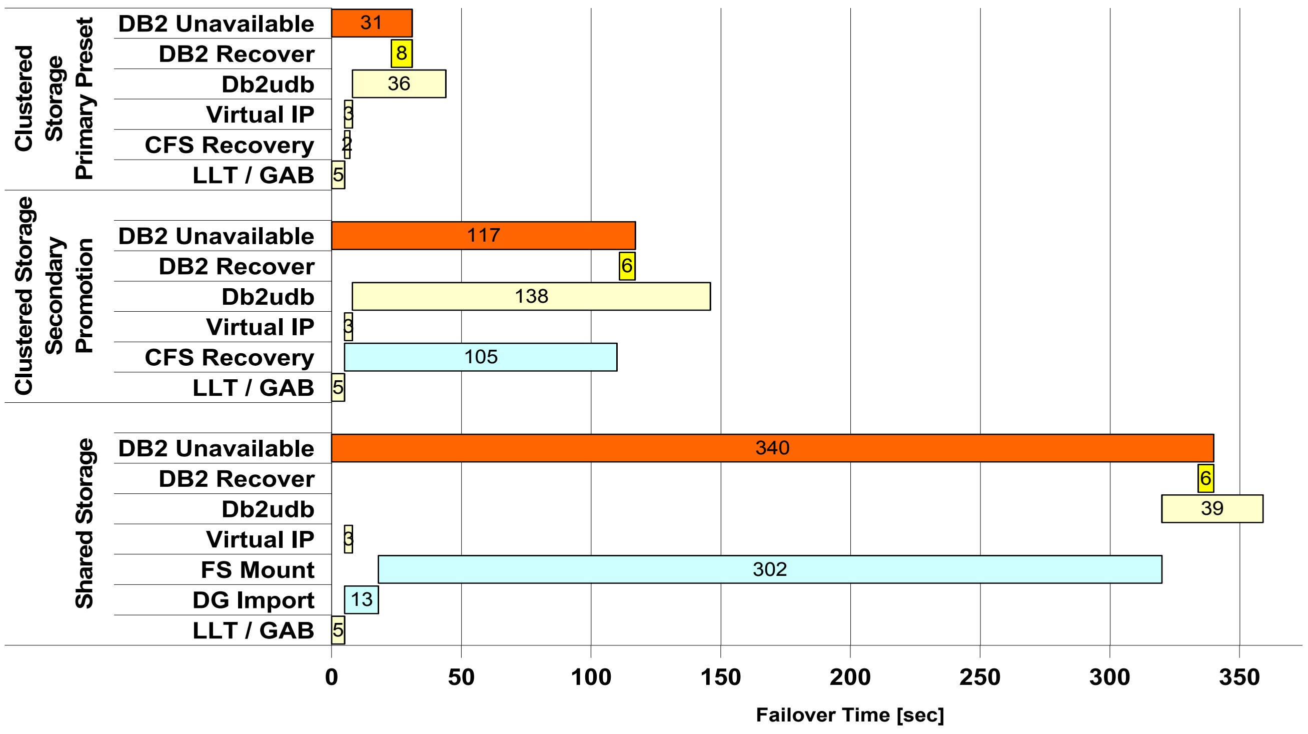


Figure 3.9 Failover Timeline for 86 Volume / File System Configuration

For a large number of file systems, the complete failover time is dominated by the file system recovery itself, unless CFS with a preset primary host is used. If the database server is acting as a CFS primary for the clustered file systems, the recovery is about one third of a conventional, failover storage configuration - 105 compared to 302 seconds, as indicated in Figure 3.9 above.

By assigning the CFS primary role to the inactive or standby server, the failover time becomes independent of the number of file systems used for the DB2 UDB database. For all of our storage configurations, the CFS failover time is less than two seconds. The total recover time is further reduced to about 1/10th of a conventional storage configuration, as illustrated in Figure 3.9 above.

During our tests we did not observe a drop in the database server performance caused by the CFS role assignments. I/O requests to data files are the same on a CFS primary or secondary. Even when the faulted server rejoined the cluster after a failover, and was reassigned the CFS primary role, no drop in performance was observed.

The only caveat of the improvement reached in our tests was the manual, script-based implementation of presetting the CFS primary. With the current version of VERITAS software products, this procedure has to be carried out by a system administrator. Using VERITAS Cluster Server trigger scripts, a fully automated assignment of CFS roles can be implemented. For the scope of this paper, we stayed with the manual preset procedure because in our two-node server configuration this could be a covered by a simple script.

Even when omitting the CFS primary preset, the high availability of the database is not jeopardized, but it will result in a longer recovery time.

3.8 DB2 UDB PERFORMANCE TEST RESULTS

During our tests, we also tracked some performance numbers using the DB2 UDB Monitor to ensure we did not see any significant performance differences when running in a CVM/CFS versus a VM/FS configuration. The charts below indicate that, on average, there is no significant performance difference between the two configurations. It is important to note that we did not spend any time tuning DB2 UDB parameters to obtain the best possible performance numbers for this test because that was not the purpose of these tests. Failover tests were attempted once every 10 minutes. They are not reflected in the charts below. See section 3.7 Failover time Test results.

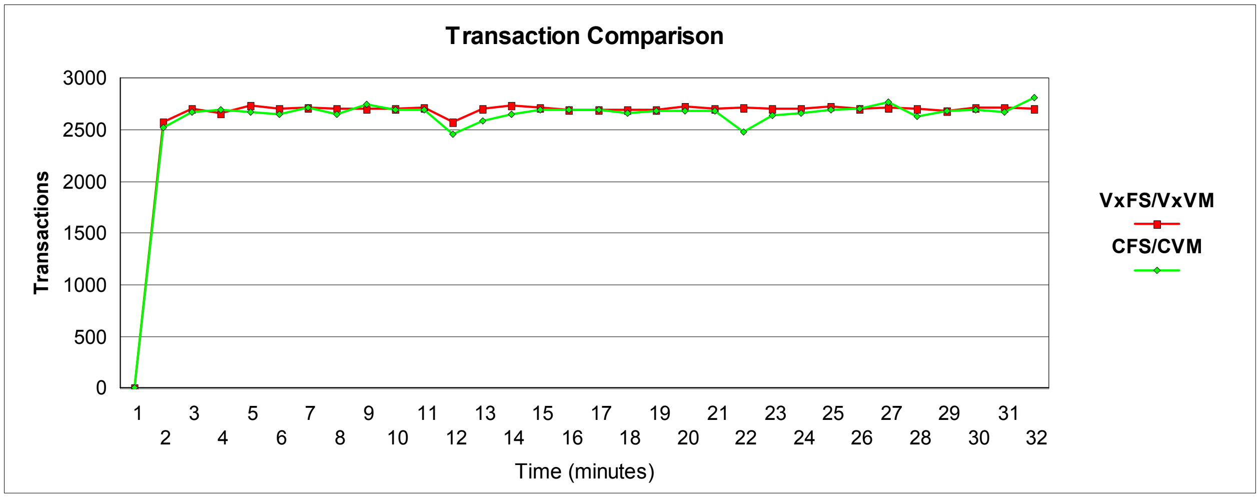


Figure 3.10 Transaction Comparisons between Clustered and Non-Clustered Environment

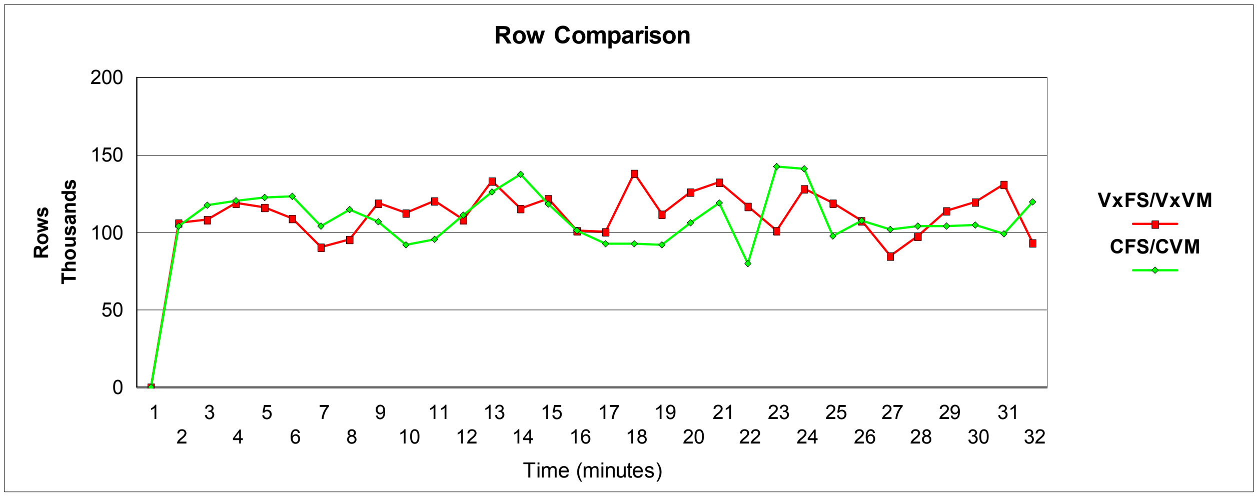


Figure 3.11 Comparison of the Rows Accessed by SQL operation in Clustered and Non-Clustered Environment

It is important to note that because of the random nature of the workload that was executed (see description earlier in section 3 of this document) the fluctuations seen in the Row Comparison chart (numbers of rows affected by SQL operations in that time interval) are to be expected. In other words, the number of rows returned will depend on which SQL statements were randomly selected to run in the one-minute time interval. This still does not skew the data and still indicates the same performance for the two environments.

The complete database manager and database configuration parameter settings used for this test can be found in Appendix B: Database MANAGER Configuration and Appendix C: Database Configuration of this document.

APPENDIX A: DB2 UDB INSTANCE START BEFORE DATABASE OPEN

Besides optimizing the storage layout and access model, as well as tuning database recovery and cluster communication timing, we investigated reducing the overall failover time by starting the database manager processes before the database was opened. For this configuration the standard VERITAS Cluster Server agent for DB2 UDB had to be modified and substituted with two customized resources.

One resource of operation type ***OnOnly*** was used to start the DB2 UDB instance, while the other of type ***OnOff*** was used to catalog and open the database. An ***OnOnly*** resource can be online on all nodes at any time, though all DB2 UDB database manager processes can be started and monitored as soon as the Cluster Server software is activated.

The "db2 start dbm" command was implemented via a rarely used Cluster Server entry point called ***open.*** The associated script is executed once at the Cluster Server daemon startup. The agent then used ***monitor*** to examine the current state and ***clean*** to restart the DB2 UDB database manager processes in case they died on their own.

The ***OnOff*** resource was very similar to the standard Db2udbAgent used in the first part of the test. However, instead of executing a "db2start" as part of ***online***, the database was cataloged on the node and then a "db2 restart database" triggered the opening of the database. The "db2 uncatalog database absolut" was done via another ***open*** entry point and was used as a precaution to prevent a parallel open of the database in case clustered storage is used.

For the catalog database command, a new attribute "DBPath" had to be introduced because the db2 catalog database command needs this as an argument.

Using this variation of the agent, we were able to decrease the total failover time by another nine seconds. The best result we obtained for all of our storage configurations was 19 seconds. Using the CFS primary preset command (as explained in 3.7 Failover Time Test results) the failover time became independent of the number and size of the database container files.

	Membership Change Detection		Storage Recovery	Service Group Restart		DB2 UDB Recovery	
	LLT	GAB	CVM/CFS	IP	Db2udb	Detect	Complete
2 Volume / File Systems	3	5	7	8	35	12	19
86 Volume / File Systems	3	5	7	8	38	12	19

Table A.1 Failover Time with DB2 UDB Instance "Pre-start" and CFS Primary Standby

We could measure the same decrease in the database recovery for shared, non-clustered storage configurations as well. But the increase in complexity of the Cluster Server agent design makes the use of different failover time tuning techniques such as cluster file system mounts more desirable, especially since the database manager pre-start reduced the failover time by a constant value of about 10 seconds, independent of the storage model. For operating system platforms other than Solaris, the gain in failover time may be less (see ftp://ftp.software.ibm.com/software/data/pubs/papers/10sfailover.pdf for additional insight).

APPENDIX B: DATABASE MANAGER CONFIGURATION

```
Database Manager Configuration

Node type = Enterprise Server Edition with local and remote clients

 Database manager configuration release level            = 0x0a00

 CPU speed (millisec/instruction)             (CPUSPEED) = 2.729512e-06
 Communications bandwidth (MB/sec)            (COMM_BANDWIDTH) = 1.000000e+00

 Max number of concurrently active databases  (NUMDB) = 8
 Data Links support                           (DATALINKS) = NO
 Federated Database System Support            (FEDERATED) = NO
 Transaction processor monitor name           (TP_MON_NAME) =

 Default charge-back account                  (DFT_ACCOUNT_STR) =

 Java Development Kit installation path       (JDK_PATH) = /usr/j2se

 Diagnostic error capture level               (DIAGLEVEL) = 3
 Notify Level                                 (NOTIFYLEVEL) = 3
 Diagnostic data directory path               (DIAGPATH) =

 Default database monitor switches
   Buffer pool                                (DFT_MON_BUFPOOL) = OFF
   Lock                                       (DFT_MON_LOCK) = OFF
   Sort                                       (DFT_MON_SORT) = OFF
   Statement                                  (DFT_MON_STMT) = OFF
   Table                                      (DFT_MON_TABLE) = OFF
   Timestamp                                  (DFT_MON_TIMESTAMP) = OFF
   Unit of work                               (DFT_MON_UOW) = OFF
 Monitor health of instance and databases     (HEALTH_MON) = OFF

 SYSADM group name                            (SYSADM_GROUP) =
 SYSCTRL group name                           (SYSCTRL_GROUP) =
 SYSMAINT group name                          (SYSMAINT_GROUP) =

 Database manager authentication              (AUTHENTICATION) = SERVER
 Cataloging allowed without authority         (CATALOG_NOAUTH) = NO
 Trust all clients                            (TRUST_ALLCLNTS) = YES
 Trusted client authentication                (TRUST_CLNTAUTH) = CLIENT
 Bypass federated authentication              (FED_NOAUTH) = NO

 Default database path                        (DFTDBPATH) = /export/home/svtdbm

 Database monitor heap size (4KB)             (MON_HEAP_SZ) = 90
 Java Virtual Machine heap size (4KB)         (JAVA_HEAP_SZ) = 1024
 Audit buffer size (4KB)                      (AUDIT_BUF_SZ) = 0
 Size of instance shared memory (4KB)         (INSTANCE_MEMORY) = AUTOMATIC
 Backup buffer default size (4KB)             (BACKBUFSZ) = 1024
 Restore buffer default size (4KB)            (RESTBUFSZ) = 1024
```

```
Sort heap threshold (4KB)                   (SHEAPTHRES) = 20000

Directory cache support                     (DIR_CACHE) = YES

Application support layer heap size (4KB)   (ASLHEAPSZ) = 15
Max requester I/O block size (bytes)        (RQRIOBLK) = 32767
Query heap size (4KB)                       (QUERY_HEAP_SZ) = 1000
DRDA services heap size (4KB)               (DRDA_HEAP_SZ) = 128

Workload impact by throttled utilities      (UTIL_IMPACT_LIM) = 100

Priority of agents                          (AGENTPRI) = SYSTEM
Max number of existing agents               (MAXAGENTS) = 2000
Agent pool size                             (NUM_POOLAGENTS) = 160
Initial number of agents in pool            (NUM_INITAGENTS) = 0
Max number of coordinating agents           (MAX_COORDAGENTS) = (MAXAGENTS -
                                            NUM_INITAGENTS)
Max no. of concurrent coordinating agents   (MAXCAGENTS) = MAX_COORDAGENTS
Max number of client connections            (MAX_CONNECTIONS) = MAX_COORDAGENTS

Keep fenced process                         (KEEPFENCED) = YES
Number of pooled fenced processes           (FENCED_POOL) = MAX_COORDAGENTS
Initialize fenced process with JVM          (INITFENCED_JVM) = NO
Initial number of fenced processes          (NUM_INITFENCED) = 0

Index re-creation time                      (INDEXREC) = RESTART

Transaction manager database name           (TM_DATABASE) = 1ST_CONN
Transaction resync interval (sec)           (RESYNC_INTERVAL) = 180

SPM name                                    (SPM_NAME) =
SPM log size                                (SPM_LOG_FILE_SZ) = 256
SPM resync agent limit                      (SPM_MAX_RESYNC) = 20
SPM log path                                (SPM_LOG_PATH) =

TCP/IP Service name                         (SVCENAME) = db2c_svtdbm
Discovery mode                              (DISCOVER) = SEARCH
Discover server instance                    (DISCOVER_INST) = ENABLE

Maximum query degree of parallelism         (MAX_QUERYDEGREE) = 4
Enable intra-partition parallelism          (INTRA_PARALLEL) = NO

No. of int. communication buffers (4KB)     (FCM_NUM_BUFFERS) = 4096
Node connection elapse time (sec)           (CONN_ELAPSE) = 10
Max number of node connection retries       (MAX_CONNRETRIES) = 5
Max time difference between nodes (min)     (MAX_TIME_DIFF) = 60

db2start/db2stop timeout (min)              (START_STOP_TIME) = 10
```

APPENDIX C: DATABASE CONFIGURATION

```
      Database Configuration for Database absolut

 Database configuration release level                    = 0x0a00
 Database release level                                  = 0x0a00

 Database territory                                      = US
 Database code page                                      = 819
 Database code set                                       = ISO8859-1
 Database country/region code                            = 1

 Dynamic SQL Query management                            (DYN_QUERY_MGMT) = DISABLE

 Discovery support for this database                     (DISCOVER_DB) = ENABLE

 Default query optimization class                        (DFT_QUERYOPT) = 5
 Degree of parallelism                                   (DFT_DEGREE) = 1
 Continue upon arithmetic exceptions                     (DFT_SQLMATHWARN) = NO
 Default refresh age                                     (DFT_REFRESH_AGE) = 0
 Number of frequent values retained                      (NUM_FREQVALUES) = 10
 Number of quantiles retained                            (NUM_QUANTILES) = 20

 Backup pending                                          = NO

 Database is consistent                                  = YES
 Rollforward pending                                     = NO
 Restore pending                                         = NO

 Multi-page file allocation enabled                      = NO

 Log retain for recovery status                          = NO
 User exit for logging status                            = NO

 Data Links Token Expiry Interval (sec)                  (DL_EXPINT) = 60
 Data Links Write Token Init Expiry Intvl                (DL_WT_IEXPINT) = 60
 Data Links Number of Copies                             (DL_NUM_COPIES) = 1
 Data Links Time after Drop (days)                       (DL_TIME_DROP) = 1
 Data Links Token in Uppercase                           (DL_UPPER) = NO
 Data Links Token Algorithm                              (DL_TOKEN) = MAC0

 Database heap (4KB)                                     (DBHEAP) = 5000
 Size of database shared memory (4KB)                    (DATABASE_MEMORY) = AUTOMATIC
 Catalog cache size (4KB)                                (CATALOGCACHE_SZ) = (MAXAPPLS*4)
 Log buffer size (4KB)                                   (LOGBUFSZ) = 4
 Utilities heap size (4KB)                               (UTIL_HEAP_SZ) = 5000
 Buffer pool size (pages)                                (BUFFPAGE) = 1000
 Extended storage segments size (4KB)                    (ESTORE_SEG_SZ) = 16000
 Number of extended storage segments                     (NUM_ESTORE_SEGS) = 0
 Max storage for lock list (4KB)                         (LOCKLIST) = 2000

 Max size of appl. group mem set (4KB)                   (APPGROUP_MEM_SZ) = 8192
```

```
Percent of mem for appl. group heap            (GROUPHEAP_RATIO) = 70
Max appl. control heap size (4KB)              (APP_CTL_HEAP_SZ) = 1024

Sort heap thres for shared sorts (4KB)         (SHEAPTHRES_SHR) = (SHEAPTHRES)
Sort list heap (4KB)                           (SORTHEAP) = 256
SQL statement heap (4KB)                       (STMTHEAP) = 8096
Default application heap (4KB)                 (APPLHEAPSZ) = 1024
Package cache size (4KB)                       (PCKCACHESZ) = 4096
Statistics heap size (4KB)                     (STAT_HEAP_SZ) = 43804

Interval for checking deadlock (ms)            (DLCHKTIME) = 6000
Percent. of lock lists per application         (MAXLOCKS) = 70
Lock timeout (sec)                             (LOCKTIMEOUT) = 120

Changed pages threshold                        (CHNGPGS_THRESH) = 20
Number of asynchronous page cleaners           (NUM_IOCLEANERS) = 20
Number of I/O servers                          (NUM_IOSERVERS) = 10
Index sort flag                                (INDEXSORT) = YES
Sequential detect flag                         (SEQDETECT) = YES
Default prefetch size (pages)                  (DFT_PREFETCH_SZ) = 32

Track modified pages                           (TRACKMOD) = OFF

Default number of containers                   = 1
Default tablespace extentsize (pages)          (DFT_EXTENT_SZ) = 32

Max number of active applications              (MAXAPPLS) = 200
Average number of active applications          (AVG_APPLS) = 32
Max DB files open per application              (MAXFILOP) = 64

Log file size (4KB)                            (LOGFILSIZ) = 500
Number of primary log files                    (LOGPRIMARY) = 250
Number of secondary log files                  (LOGSECOND) = 0
Changed path to log files                      (NEWLOGPATH) =
Path to log files                              = /db03/svtdbm/NODE0000/SQL00001/SQLOGDIR/
Overflow log path                              (OVERFLOWLOGPATH) =
Mirror log path                                (MIRRORLOGPATH) =
First active log file                           =
Block log on disk full                         (BLK_LOG_DSK_FUL) = NO
Percent of max active log space by transaction (MAX_LOG) = 0
Num. of active log files for 1 active UOW      (NUM_LOG_SPAN) = 0

Group commit count                             (MINCOMMIT) = 1
Percent log file reclaimed before soft chckpt  (SOFTMAX) = 1
Log retain for recovery enabled                (LOGRETAIN) = OFF
User exit for logging enabled                  (USEREXIT) = OFF

Auto restart enabled                           (AUTORESTART) = ON
Index re-creation time                         (INDEXREC) = ACCESS
Default number of loadrec sessions             (DFT_LOADREC_SES) = 1
Number of database backups to retain           (NUM_DB_BACKUPS) = 12
Recovery history retention (days)              (REC_HIS_RETENTN) = 366

TSM management class                           (TSM_MGMTCLASS) =
```

TSM node name	(TSM_NODENAME) =
TSM owner	(TSM_OWNER) =
TSM password	(TSM_PASSWORD) =

APPENDIX D: VERITAS CLUSTER SERVER CONFIGURATION FILES

The following paragraph is a copy of the VERITAS Cluster Server main.cf file for a two-volume / file system configuration as discussed in 3.2 Baseline Cluster Configuration for Disk group Failover. The only difference for the 14 and 86 volume / file system setup is the number of Mount resources. For the larger configuration, the "datashark" resource was replaced by an appropriate number of Mount resources linked to resource "absolut_udb" of type Db2udb and sun-ha3bshark of type IP.

```
include "types.cf"
include "Db2udbTypes.cf"

cluster Sun_CVM_Test (
      UserNames = { admin = "W074SGNO6c51." }
      Administrators = { admin }
      CounterInterval = 5
      )

system sun-ha3 (
      )

system sun-ha4 (
      )

group db2_shark (
      SystemList = { sun-ha3 = 1, sun-ha4 = 2 }
      AutoStartList = { sun-ha4 }
      )

      Db2udb absolut_udb (
            DB2InstOwner = svtdbm
            DB2InstHome = "/export/home/svtdbm"
            IndepthMonitor = 1
            DatabaseName = absolut
            )

      DiskGroup db2shark (
            DiskGroup = db2shark
            )

      IP sun-ha3bshark (
            Device = hme0
            Address = "10.0.35.128"
            NetMask = "255.255.255.0"
            )

      Mount datashark (
            MountPoint = "/db03/data"
            BlockDevice = "/dev/vx/dsk/db2shark/datashark"
            FSType = vxfs
            MountOpt = "largefiles,qio"
            FsckOpt = "-y"
            )
      Mount db03shark (
            MountPoint = "/db03/svtdbm"
            BlockDevice = "/dev/vx/dsk/db2shark/db03shark"
            FSType = vxfs
            MountOpt = "largefiles,qio"
            FsckOpt = "-y"
            )

      NIC hme0shark (
            Device = hme0
            NetworkType = ether
            )

      absolut_udb requires datashark
      absolut_udb requires db03shark
      absolut_udb requires sun-ha3bshark
      datashark requires db2shark
      db03shark requires db2shark
      sun-ha3bshark requires hme0shark


      // resource dependency tree
      //
      //    group db2_shark
      //    {
      //    Db2udb absolut_udb
      //        {
      //        IP sun-ha3bshark
      //            {
      //            NIC hme0shark
      //            }
      //        Mount db03shark
      //            {
      //            DiskGroup db2shark
      //            }
      //        Mount datashark
      //            {
      //            DiskGroup db2shark
      //            }
      //        }
      //    }
```

The section is the main.cf file for a two-volume clustered configuration. For the 14 and 86 volume setups, the CFSMount datash_fs resource was replaced by a matching number of CFSMount resources linked to the CVMDiskGroup resource.

```
include "types.cf"
include "CFSTypes.cf"
include "CVMTypes.cf"
include "Db2udbTypes.cf"

cluster Sun_CVM_Test (
      UserNames = { admin = "<admin_password>" }
      Administrators = { admin }
      CounterInterval = 5
      )

system sun-ha3 (
      )

system sun-ha4 (
      )

group cvm (
      SystemList = { sun-ha3 = 0, sun-ha4 = 1 }
      AutoFailOver = 0
      Parallel = 1
      AutoStartList = { sun-ha3, sun-ha4 }
      )

      CFSQlogckd qlogckd (
            Critical = 0
            )

      CFSfsckd vxfsckd (
            ActivationMode @sun-ha3 = { db2dg1sh = sw }
            ActivationMode @sun-ha4 = { db2dg1sh = sw }
            )

      CVMCluster cvm_clus (
            Critical = 0
            CVMClustName = vcs
            CVMNodeId = { sun-ha3 = 3, sun-ha4 = 4 }
            CVMTransport = gab
            CVMTimeout = 200
            )

      vxfsckd requires qlogckd


      // resource dependency tree
      //
      //    group cvm
      //    {
      //    CVMCluster cvm_clus
      //    CFSfsckd vxfsckd
      //        {
      //        CFSQlogckd qlogckd
      //        }
      //    }


group db2_group1sh (
      SystemList = { sun-ha3 = 1, sun-ha4 = 2 }
      AutoStartList = { sun-ha3 }
      )

      Db2udb absolut (
            DB2InstOwner = svtdbm
            DB2InstHome = "/export/home/svtdbm"
```

```
                IndepthMonitor = 1
                DatabaseName = absolut
                )

        IP sun-ha3bsh (
                Device = hme0
                Address = "10.0.35.128"
                NetMask = "255.255.255.0"
                )

        NIC hme0sh (
                Device = hme0
                NetworkType = ether
                )

        requires group db2dg1_shared online local firm
        absolut requires sun-ha3bsh
        sun-ha3bsh requires hme0sh


        // resource dependency tree
        //
        //      group db2_group1sh
        //      {
        //      Db2udb absolut
        //          {
        //          IP sun-ha3bsh
        //              {
        //              NIC hme0sh
        //              }
        //          }
        //      }

group db2dg1_shared (
        SystemList = { sun-ha3 = 1, sun-ha4 = 2 }
        Parallel = 1
        AutoStartList = { sun-ha3, sun-ha4 }
        )

        CFSMount datash_fs (
                MountPoint = "/db03/data"
                BlockDevice = "/dev/vx/dsk/db2shark/datashark"
                MountOpt = "largefiles,qio"
                Primary = sun-ha4
                )

        CFSMount db03sh_fs (
                MountPoint = "/db03/svtdbm"
                BlockDevice = "/dev/vx/dsk/db2shark/db03shark"
                MountOpt = "largefiles,qio"
                Primary = sun-ha4
                )

        CVMVolDg db2dg1sh (
                CVMDiskGroup = db2shark
                CVMVolume = { db03shark, datashark }
                CVMActivation @sun-ha3 = sw
                CVMActivation @sun-ha4 = sw
                )

        requires group cvm online local firm
        datash_fs requires db2dg1sh
        db03sh_fs requires db2dg1sh


        // resource dependency tree
        //
        //      group db2dg1_shared
        //      {
        //      CFSMount datash_fs
        //          {
        //          CVMVolDg db2dg1sh
        //          }
        //      CFSMount db03sh_fs
        //          {
        //          CVMVolDg db2dg1sh
        //          }
        //      }
```