**Integriertes Seminar Datenbanken und Informationssysteme**

**im Wintersemester 2005/2006**

**Dependable Adaptive Information Systems (DAIS)**

# Highly Available Database Systems

**Advised by: Prof. Dr.-Ing. Dr. h. c. Theo Härder**

**Author: Ou Yi**

**13 Jan 2006**

# Table of Contents

# Highly Available Database Systems

Ou Yi

Fachbereich Informatik
Technische Universität Kaiserslautern
o_yi@informatik.uni-kl.de

**Abstract.** In this article, we present the technologies for achieving high availability of database systems. First we introduce mainframes and virtual machines as possibly reliable platforms. Then, based on distributed and parallel database systems, we discuss the concepts and architectures of highly available database systems such as shared everything, shared nothing, and shared disk. Following that we give an overview for the clustering technology and other high availability technologies used in the commercial database systems.

## 1 Introduction

“Build a system used by millions of people that is always available – out less than one second per 100 years.” -- J. Gray [9]

Today, we depend so heavily on information systems that system outage or loss of data is more and more intolerable. Sometimes the loss of critical data relates directly to the survivability of an enterprise. This draws our attention to the dependability of information systems. According to [8], *dependability* is “the ability to avoid service failures that are more frequent and more severe than is acceptable” and *availability* is one of the most important attributes of dependability. Since database systems are the backend of most information systems, their availability is critical to the availability of whole information system.

*Availability* (A) can be defined as: the fraction of the offered load that is processed with acceptable response times [12]. This definition allows a system to be counted for available, even if some parts of it are being not available (e.g. being repaired or maintained), as long as the promised service load is achieved. The requirement on “acceptable response time” means that availability can not be pursued alone without consideration of performance. Statistically, availability can be quantified as:

$$A = MTTF / (MTTF + MTTR) \qquad (1)$$

MTTF (Mean Time to Failure) is the average time that a system runs (without failing) after it has been set up or repaired. MTTR (Mean Time to Repair) is the average time needed to repair (or recover) a failed system [10].

Availability is often expressed in number of nines as well. The calculation of this number is based on Equation (1), with the denominator MTTF+MTTR equals one year. See **Table 1**.

**Table 1.** Availability expressed in number of nines

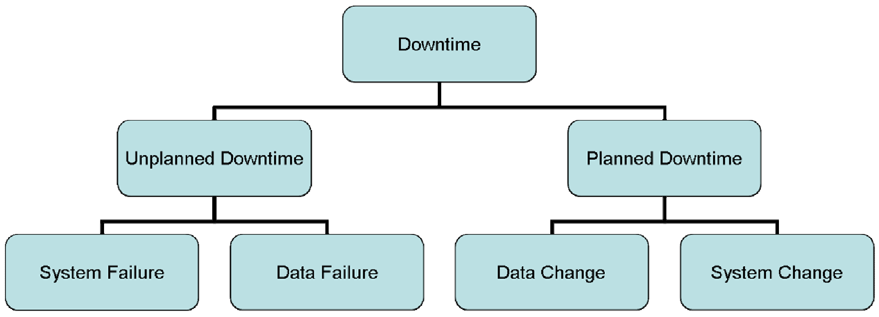
| System Type | Availability | Unavailability (min/year) |
|---|---|---|
| Well-managed | 99.9 % | 526 |
| Fault-tolerant | 99.99 % | 53 |
| High-availability | 99.999 % | 5 |
| Very-high-availability | 99.9999 % | 0.5 |
| Ultra-availability | 99.99999 % | 0.05 |

*High Availability* (HA) refers to the availability of a system which approaches one hundred percent. From Equation (1) we know: to achieve A = 1 approximately, we have to either maximize MTTF to the utmost so that MTTF approaches ∞ ad infinitum, or minimize MTTR so that it is small enough and can be ignored compared with MTTF. The latter is the task of ROC (Recovery Oriented Computing), so we focus our observation in this article more on the former aspect, i.e. the maximization of MTTF. But it's worth mentioning that sometimes these two aspects are closely related to each other.

Each system has an ideal specified behavior and an observed behavior. A *failure* occurs when the observed behavior deviates from the specified behavior. A failure occurs because of an *error*. The cause of the error is a *fault*. Generally speaking, using components which are reliable and fault tolerant, isolating errors, so that they will not be propagated, and using redundancy for less reliable software or hardware components, can help to extend MTTF and thus improve the overall system availability.

In a production environment, to achieve high availability means to eliminate downtime. For this purpose, both unplanned downtime and planned downtime must be addressed. *Unplanned downtime* is caused primarily by computer failure or data failure. *Planned downtime* is primarily due to data changes or system changes, for example, server maintenance or upgrade of operating system. See Figure 1.

**Fig. 1.** Causes of Downtime



The following chapters are structured this way: in chapter 2 we talk about the platform for highly available DBS. Chapter 3 and 4 are the theoretical fundamentals of highly available DBS: distributed and parallel DBS. In chapter 5 we introduce clustering technology. In chapter 6 we give an overview of the HA features of commercial DBS products.

# 2 Platform for highly available DBS

A *database system* (DBS) basically consists of databases and a database management system (DBMS). The DBMS is a piece of software which runs on top of the operating system or sometimes acts as an extension of the operating system [4]. In both cases, either directly or indirectly, the DBMS depends on the underlying hardware (e.g. architecture, instruction set, etc.), i.e. the platform. Virtual machines can also be viewed as a kind of platform. We will talk about them later in this chapter.

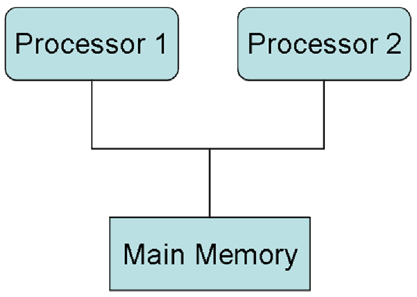
## 2.1 Hardware Architecture

The most mission critical database systems are also at the same time heavily loaded, e.g. they are executing thousands of transactions per second. This kind of tasks is today even for the most powerful single processor computer “mission-impossible”. They can only be fulfilled by the cooperation of multiple computers. In a multi-computer system, processors are coupled together through network connection or through hardware, e.g. shared cache or shared memory, so that they can communicate with each other. *Tight coupling*, *loose coupling*, and *close coupling* are typical types of coupling.

### 2.1.1 Tight Coupling

In a tightly coupled system, a single main memory is shared between the processors. Figure 2 shows a tightly coupled system consisting of two processors. Normally, the software, e.g. operating system or applications, can have only one instance in such a system. *Multiprocessor* is often used as synonym of tight coupling. *Symmetrical Multiprocessor* (SMP) is the most common type of multiprocessor, where two or more identical processors are connected to a single shared main memory.

**Fig. 2.** Tight Coupling



The advantage of tight coupling is its higher computing power compared with that of a single processor computer, because instructions can be executed in parallel by multiple processors. The communications between these processors are generally efficient, because they all have direct access to the shared main memory. Traditionally, tight coupling is criticized of having low availability and poor extensibility. It has low availability, because errors can not be effectively isolated with a single main memory. Extensibility refers to the capability of adding new processors. As the number of processors being increased in the system, the single main memory becomes a bottleneck. Larger caches can help to reduce the number of memory accesses but cache coherency control causes also more overhead. Therefore the number of processors in a multiprocessor has an upper limit. However, as we will see later in this chapter, these problems are addressed with the modern technology.

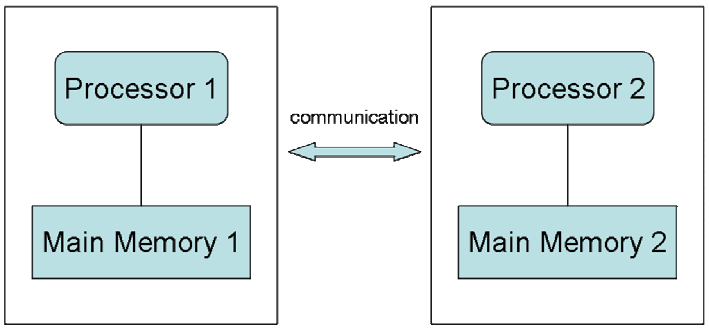
### 2.1.2 Loose Coupling

A loosely coupled system is a cluster of independent computers, interconnected through a network. In this system, each processor has its own main memory, to which only the owner has direct access. Therefore each computer has its own copy of operating system and application, e.g. a DBMS[1] instance, running in its private main memory. This leads to better error isolation. Due to the independence, new computer can be easily added into the system. However, the communication, even over a fast

[1] DBMS can be viewed as server application.

network, is much more expensive compared with data exchange using a shared memory as in the case of tight coupling. Figure 3 shows a loosely coupled system consisting of two computers.

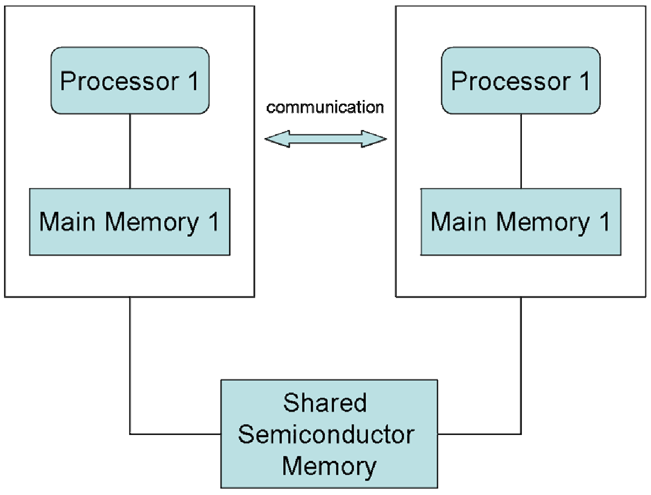
**Fig. 3.** Loose Coupling



### 2.1.3 Close Coupling

The goal of close coupling is to overcome the disadvantage of loose coupling, i.e. high communication cost, without loosing its advantages. In a closely coupled system, each computer also owns a private main memory and private copy of software. In addition, there is a coupling component in the system. It is normally a semiconductor memory, to which all processors in the system have direct access. High-speed data exchange between the processors can be realized using this coupling component. See Figure 4.

**Fig. 4.** Close Coupling



### 2.1.4 An Example

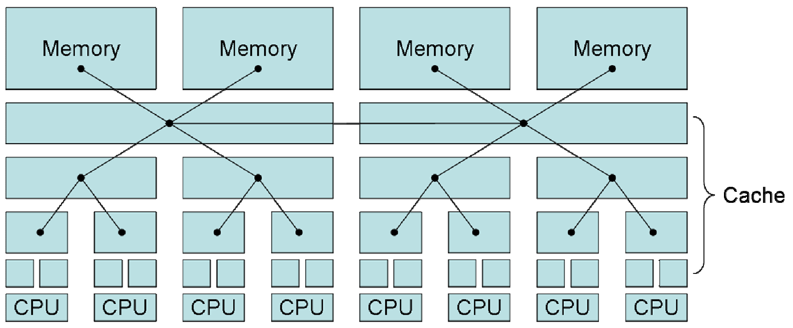
Mainframes (super-computers), which were once in time believed to be "dead", experienced a renaissance in the last decade, due to their high availability and high performance, which are essential for our discussion. As an example, we will talk about the mainframe products zSeries and System z9 from IBM.

The IBM product families, zSeries ("z" means zero downtime) and System z9, both succeeded the IBM S/390 and support the S/390 instruction sets. The most remarkable characteristic of z/Series and System z9 is the so-called *symmetrical memory-coherent multiprocessor* structure[2]. Among them, the newest server model, System z9 109 [13], can have up to 54 configurable processor units (PU: comparable with processor).

*Memory-coherent SMP*

A zSeries server is basically an SMP with a large shared level 2 cache (L2). The large shared L2 requires special design and production techniques, but it helps to solve the problems of *non-uniform memory access* (NUMA). NUMA is a kind of memory design used in multiprocessors, where the memory access time depends on the memory location relative to a processor (see Figure 5).

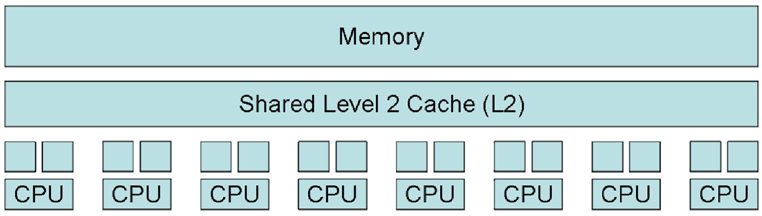
**Fig. 5.** Non-Uniform Memory Access



Under NUMA, a processor can access its own local memory faster than non-local memory, i.e. memory which is local to another processor or shared between processors, due to the cache hierarchy and inter-processor communication. Furthermore, maintaining cache coherence across shared memory has a significant overhead. These characteristics impact the performance and scalability of the multiprocessor. In zSeries' design, there is only one large L2, which is shared between all PUs. Thus, access time to memory is uniform to all PUs and the L2 cache is always coherent (See Figure 6).

[2] From now on, we may make no difference between z/Series and System z9 for convenience, since they have similar architecture.

**Fig. 6.** Memory-coherent SMP



*Modular Multi-book Design*

New zSeries servers such as the z9-109 are built using a modular multi-book design that supports one to four *books* per server. A book contains a multi-chip module (MCM), which hosts the PUs (12 or 16 per book), memory, and high speed connectors for I/O. This kind of design enables many of the high-availability, non-disruptive features such as the Concurrent Book Add (CBA) and the Enhanced Book Availability (EBA). CBA allows a server to be concurrently upgraded by adding new books without affecting application processing. EBA allows a single book, in a multi-book server, to be concurrently removed from the server and reinstalled during an upgrade or repair action, provided that the remaining books have sufficient physical resources to take over the load. The I/O resources connected to the removed book can still be accessed by the remaining books through redundant I/O interconnects.

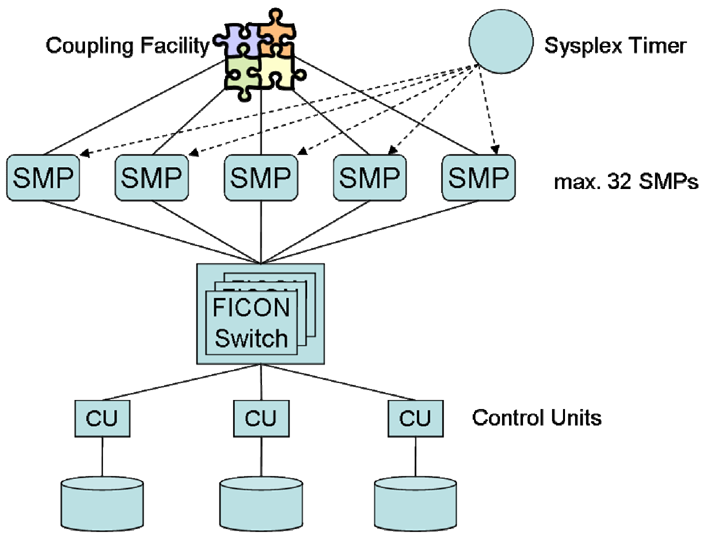
*PU Sparing and Instruction Retry*

PUs can play different and specialized roles, e.g. PUs responsible for processing, PUs responsible for I/O, etc. On each z9-109 server, two PUs are reserved as spares. If a PU fails, its role is dynamically and transparently reassigned to a spare PU. To detect errors in a PU, each instruction is executed two times, and then the results will be compared. If the results differ from each other, an error is detected and the execution will be retried (Instruction Retry). If the initial error is a soft error (intermittent), the retries will succeed. If the number of retries exceeds a threshold, the PU is believed to be failed and a target spare PU will be selected to take over the execution.

*Close Coupling*

The IBM Parallel Sysplex Cluster is a typical example of close coupling. A Parallel Sysplex allows up to 32 SMPs (nodes) to work together in a cluster and share the same external storage [3]. The nodes and the external storage are all connected to a group of FICON (Fiber Connection) switches. Thus each node can directly access all the disks and communicate with all the other nodes through a fast link.

**Fig. 7.** Parallel Sysplex



The key component of a Parallel Sysplex is the so-called Coupling Facility (CF), which is a specialized SMP with a large global main memory (measured in Gigabytes). It is global because it can be accessed by all other nodes in the cluster through very fast fiber optic links (100 Mbytes/s). *This global memory is the symbol of close coupling and of the data sharing architecture.*[3] The access to this global memory is synchronous, i.e. without context switch. CF is useful for the global control tasks, such as concurrency control and buffer coherency control for DBMSs. To remove a *single point of failure* (SPOF), two or more CFs are recommended to be used in a Parallel Sysplex [15]. This also improves performance.

Another shared component in a Parallel Sysplex is the Sysplex Timer, which provides other nodes with a global unique time. The Sysplex Timer is useful for database systems such as DB2 and IMS. They need this for recovery. Because the external storage is shared, any data object can be accessed and updated on multiple nodes. These updates are sequence-dependent, e.g. a later update depends on the result of an earlier update. With the global unique time, all the changes made to a data object can be sorted according to their timestamp and recorded in a global log file. So after a failure, the changes to a data object made on multiple nodes can be completely and correctly reconstructed [1]. Similar to the CF, two Sysplex timers are recommended for availability reasons.

[3] Data sharing architecture will be introduced in section 4.4.

To effectively support communication between the nodes, access to external storage and access to CF, special machine instructions and OS services are provided by the IBM z/Series and its operating system z/OS. These services are used primarily by software subsystems such as database systems. A concrete example is DB2 for OS/390, which uses CF for global locking and buffer coherency control.

## 2.2 Virtual Machine

A Virtual Machine is a fully protected and isolated copy of the underlying physical machine's hardware. Thus, each virtual machine user is given the illusion of having a dedicated physical machine [2]. The core of this technology is the *Hypervisor*. Hypervisor is a kind of software which emulates the underlying physical machine's architecture very efficiently by directly using the machine code. Using this technology, we can have multiple virtual machines on a single physical server. The virtual machines run in their own address spaces and are independent from each other. Each virtual machine can act as a stand-alone server and have his own operating system and applications. The operating system of the virtual server is called *guest* operating system. The Hypervisor is called *host* operating system.

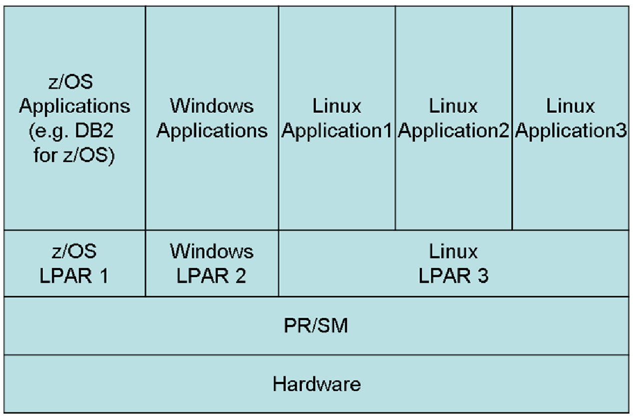
This brings us many advantages: 1. easier administration due to server consolidation; 2. better security, if we consider that each virtual server forms a boundary for security policy; 3. the resources owned by virtual machines can be easier reallocated than on physical machines, thus the resources can be better utilized and this also makes the scalability of the virtual server better; 4. overcoming the hardware boundaries; 5. error isolation; 6. makes high-availability solutions more flexible. In the following sections, we will talk about point 4 to 6 in more details.

### 2.2.1 Overcoming Hardware Boundaries

Modern enterprise servers are normally equipped with a large number of CPUs and lots of other physical resources. Virtual Machine technology makes them more powerful with the ability of optimally allocating the resources.

As example, we look at the IBM zSeries again. In a zSeries server, all books are interconnected with a very high-speed internal communication links via the L2 cache, which allows the books altogether to be logically viewed as a large symmetrical, memory-coherent multiprocessor, with up to 54 PUs in the case of z9-109. This logical view is maintained and managed by the PR/SM facility. PR/SM (Processor Resource / System Manager) is a Hypervisor. It allows the large SMP to be partitioned in multiple virtual machines. Each of them runs in a logical partition (LPAR). Figure 8 shows three LPARs managed by a PR/SM, having different operating systems.

**Fig. 8.** PR/SM and LPAR



An LPAR has the following resources:

- one or several CPUs, or slices of CPU time in case of time sharing
- a zone, which is a part of the physical main memory
- channels, which can be viewed as I/O resources

Among them, only channels are statically assigned to LPARs. The CPUs and the main memory can be reassigned to LPARs on the fly. The PR/SM for z9-109 has the ability to configure and operate as many as 60 LPARs, which have processors, memory, and I/O resources assigned from any of the installed books.

PR/SM is not the only product on the market, which supports server virtualization. Another product from IBM, z/VM, has similar functions. z/VM is an operating system for the z/Series architecture. It is in fact a Hypervisor extended with some operating system functions. Furthermore, EMC’s VMware and Microsoft’s VirtualPC are well-known products, which emulate other hardware architectures, e.g. IA32.

### 2.2.2 Error Isolation

If a particular guest machine crashes, the Hypervisor guarantees that there is zero impact to any other part of the host. So for example, we can use a dedicated virtual machine for certain application, for which occasional crashes are acceptable but the crashes should not impact the other more critical applications. The dedicated virtual machine can be so configured, that it only consumes the resources which are necessary for this application. This is much more flexible and economical than using another physical server.

### 2.2.3 More Flexible HA Solutions

*Many to One*

A common practice to achieve high availability is to have a standby server in sync with the production server so that, if there is a failure on the production server, the standby server can take over the processing. Some servers don't coexist on a single physical machine. For example, we have a DB2 on a Linux machine and an SQL Server on a Windows machine. To protect both servers, we have to use two physical standby servers. Now with virtual machine technology, we can configure two virtual servers as standby running on a single physical server. This many-to-one approach provides for large enterprises which have thousands applications great flexibility and cost advantages in deploying their high-availability solutions.

*Another Granularity of Replication / Backup*

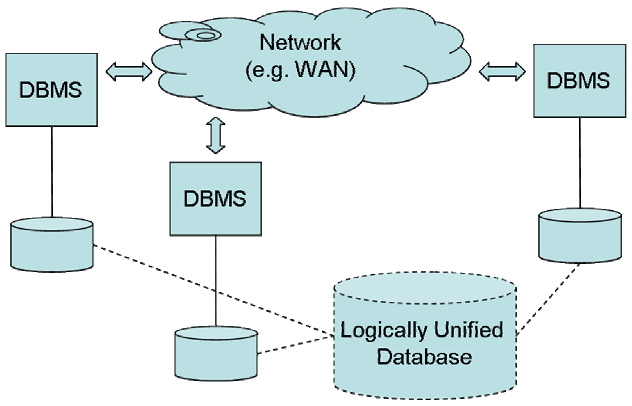
The hard disk of a guest machine is often realized as a file on the host machine. For some server applications, it is acceptable to just backup the disk. So replicating the file on the host machine equals having a mirrored disk of the virtual server. This again means greater flexibility in choosing replication and backup strategies.

## 3 Distributed Database Systems

A *distributed database* is a collection of multiple, logically interrelated databases distributed over a computer network [7]. A *distributed database management system* (distributed DBMS) is the software system which manages the distributed database and makes the distribution transparent to the users. In a *distributed database system* (DDBS), each node runs an instance of the DBMS, which manages a part of the distributed database, i.e. the local database. The DBMS instances cooperate and communicate with each other, possibly across sites and over a wide area network (WAN). The following figure shows a DDBS.

Even if the database is physically distributed over multiple nodes and processed by multiple DBMS instances, the user or the database application is provided with a logically integrated view of the database. This is also called "distribution transparency". Maintaining the distribution transparency means: an application written for a centralized DBS doesn't need to be rewritten to run in the distributed environment. This is quite a challenge for the implementation of DDBS due to lack of global information.

**Fig. 9.** Distributed Database System



DDBS and PDBS (parallel DBS, see next chapter) both involve multiple computers. They are not only intended to improve performance, but also to achieve high availability, since they have replicated components and thus potentially eliminate single points of failure. In addition, the computing power of the overall system can be improved by adding new computers or nodes into the system. Management and maintenance work (e.g. installation of operating system service packs) can be accomplished in a rolling style, e.g. node by node, without taking the whole system down, which is a must in single-node systems, thus resulting in reduced downtime of the whole system.

## 3.1 Distributed Commitment Protocols

In a DDBS, four types of failures are possible: transaction failures, system failures, disk failures and communication line failures. The first three types of failure are common also in centralized DBS. Transactions can fail due to problematic input data, deadlocks, etc. A system failure is a system crash which leads to loss of main memory contents. Disk failure means that data or log files saved on a disk can not be accessed any more. Communication line failures are unique to DDBS. Distributed commitment protocols have to deal with all these problems to maintain the atomicity (A) and durability (D) of transactions.

The most popular commitment protocol is distributed two-phase commit. Distributed two-phase commit is well explained in many publications, e.g. [1], so we just talk about it here briefly, because it is at least related to the disaster recovery which we will talk about in section 3.4. In a DDBS, transactions are extended to multiple nodes, i.e. a transaction may consist of several sub-transactions which are executed on multiple computers. Generally speaking, distributed two-phase commit applies synchronous log writes for all protocol state changes to ensure stable protocol decisions and finally the durability (D) of transactions. To guarantee the atomicity (A) of transactions, a transaction can be committed only when all its sub-transactions are able to commit and their changes are written in log files, otherwise all changes made by this transaction and its sub-transactions are rolled back.

## 3.2 Global Serializability

Serializability is used to judge the correctness for concurrent execution of transactions. In DDBS, this criterion is extended to *Global Serializability* [7]. The concurrent execution of a set of distributed transactions is globally serializable if and only if:

1. the execution of the set of transactions at each node is serializable, and
2. the serialization orders of these transactions at all these nodes are identical.

This means, transactions having read and modified objects at multiple nodes must be serialized in the same order at all nodes. Global serializability is guaranteed by *distributed concurrency control* algorithms, which maintain the Isolation (I) property of transactions.

Primary-copy locking algorithm is useful for replicated databases, where multiple copies of data items may be stored at various nodes and may be read or updated there. One of the copies is appointed as the primary copy. In order to access any copy of a data item, a lock should be set on its primary copy. Thus all the lock requests are directed to the nodes where the primary copies are stored. The lock manager at the primary-copy node is responsible for setting and releasing the lock.

Primary-copy locking and other locking based distributed concurrency control algorithms all have the problem of deadlock, which causes further complexity by requiring global deadlock detection and prevention techniques, etc. Alternatives of the locking-based approach include timestamp-based algorithms or optimistic concurrency control [1] [11].

## 3.3 Data Replication

The data replication aspect of DDBS is related to the consistency (C) property of transactions. In a DDBS, data items can be replicated across nodes or sites, thus potentially high availability can be achieved, because a replicated data item is still accessible when a node fails and the other nodes hold the replicas. Replication can also be used to improve performance, if the replicas are allowed to be read or updated and they are located near to the nodes where they are accessed. The problem is how to maintain consistency between multiple copies of a data item if any of them can be updated.

The Read-Once/Write-All (ROWA) algorithm can be used to achieve the so-called *one-copy equivalence*, which means that all the copies of a data item are identical after any updates. One-copy equivalence is desired because it ensures consistency. ROWA works as follows: a reading transaction can read any copy of a data item (read once), while an updating transaction has to also update all the replicas as part of the same transaction. ROWA is simple and straightforward and reading transactions can be processed with flexibility. But it has two strong disadvantages: 1. poor response time, because the update should be done synchronously (in one transaction) on all nodes holding the copies; 2. low availability because, if one node holding a copy fails, the updating transaction will never terminate, and the data item becomes no longer updateable.

Voting algorithms such as the majority-consensus algorithm and the quorum voting algorithm address these problems by allowing an updating transaction to commit without all copies of a data item being updated synchronously. A quorum voting algorithm works as follows: each copy of a data item is assigned with a vote; each operation has to obtain a read quorum ($V_r$) or write quorum ($V_w$) to read or write a data item, respectively; The quorums can be chosen with consideration of load balance, but they must obey these rules:

1. $V_w > T/2$,
2. $V_r + V_w > T$,

where T is the total of votes on all copies of a data item. Rule 1 ensures that a data item will not be updated by two or more transactions at the same time, thus write-write conflicts are avoided. Similarly, with Rule 2 read-write conflicts are avoided. An operation can be carried out as soon as enough votes are gathered. After an updating operation terminates, the remaining nodes holding the copies can be updated asynchronously. For example, those updates can be piggybacked on other messages. We explain this algorithm with the following example:

Object A is replicated on four nodes: <N1, N2, N3, N4>. Two votes are assigned to N1, one vote is assigned to each of the remaining nodes, i.e.: <2, 1, 1, 1>, and the total votes T is 5. Now we have to choose the read quorum ($V_r$) and write quorum ($V_w$). We choose 3 for $V_w$, which obeys Rule 1, and 3 for $V_r$, which obeys Rule 2. To update (or read) A, at least 2 nodes will be involved (if an operation gets 2 votes from N1 and 1 vote from any of the rest 3 nodes), or 3 nodes will be involved (the operation gets 1 vote from each of N2, N3, N4). The object A is still accessible even if N1 failed, because a total of 3 votes can still be gathered from N2, N3, and N4.

The votes assigned to nodes are a kind of weighting so that a node with more votes will be statistically preferred when accessing data, as in our example N1. Therefore this algorithm is also-called weighted voting algorithm. The problems of quorum voting algorithms are: 1. it is difficult to choose suitable read quorums and write quorums without involving administrators; 2. even reading transactions have to obtain quorums. That makes the overall system output lower, because it is generally believed that reading transactions occur more often than writing transactions. These problems have limited the use of these algorithms in commercial distributed DBS products.

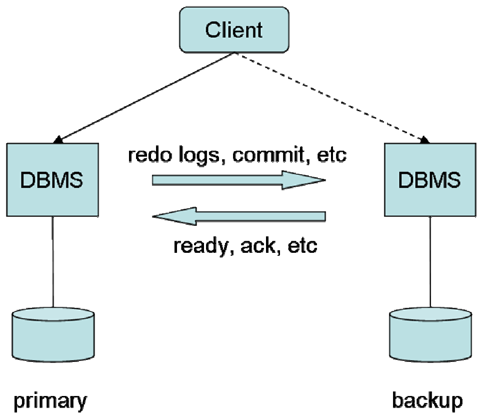
The cost of achieving strict consistency (like the one-copy equivalence) of the replicas in a DDBS is generally very high due to the network communication and the high number of messages to be exchanged. For some applications we don't need so much consistency, e.g. for some analysis and reporting tasks we only need the database to be consistent in some point of time in the past. For some other applications, the database is updated periodically or the update happens not very often. For this kind of applications, we can trade consistency for cost. Snapshots are practical for this purpose. They are semantically similar to materialized views, which provide for users or applications a consistent view of the database or tables at a point of time. Normally, snapshots can be created using SQL statement "create snapshot". With options, snapshots can be periodically refreshed or the refresh can be triggered by some events. This mechanism is supported by most commercial DBS products.

## 3.4 Disaster Recovery

A disaster, e.g. an earthquake or a terrorism attack, can destroy a complete computer center. In a centralized DBS, the only way to recover from a disaster is to retrieve the database from a backup medium, which is normally saved on magnetic tapes and kept at a remote site. The archived logs have to be replayed as well. But this kind of recovery takes a lot of time (e.g. several hours or days) and all changes made, since the last archive copy is written, will be lost. In a DDBS, it is possible to have a better solution. The complete database can be replicated online to a remote site, so that in case of a disaster the remote site can take over the processing quickly.

In the simplest case, there are a primary system which processes the workload and a backup system which manages the replica of the database. The database on the remote site can be kept up-to-date by applying redo logs of the primary database. How quickly the takeover can be done depends on how up-to-date is the database at the remote site. See Figure 10 for this configuration.

**Fig. 10.** Replicated Database for Fast Disaster Recovery



Ideally the backup system has applied all the committed changes to its database before the disaster happens. This can be ensured by applying a distributed two-phase commitment protocol which is introduced in section 3.1. So a transaction commits only if both the primary and backup are ready and both agree to commit. If one of the systems is unavailable or the connection between primary and backup is broken, no transaction can be committed. This approach is called *very safe* [12], because no committed transaction will be lost unless disaster happens at both sites. However, the availability of such a system is even worse than that of a single system. Because, if either of the two systems is down, no transactions can be committed and the probability that any of the two systems is down is almost the double probability that one system is down.

The so-called *1-safe* approach allows the primary to commit immediately after the log is written onto disk. Then the logs will be transferred to the backup asynchronously. From the primary system's point of view there is no difference to a standard two-phase commit. But the 1-safe approach risks lost transactions, because the logs transferred to the backup can be lost due to communication error. If this happened and the primary system failed, we have only a backup system which is not consistent or not up-to-date.

In a *2-safe* approach, the backup system is also involved in the commit process like in the very-safe approach, but the primary system is independent of the backup system. If the backup is declared to be down (or it is detected by the primary), the primary commits its transactions unilaterally without the backup. Otherwise, the primary will not commit until the backup responds. The backup has the option of responding immediately upon receipt of the log or responding after the log has been written onto disk. The former improves response time. The latter enhances consistency of the backup system.

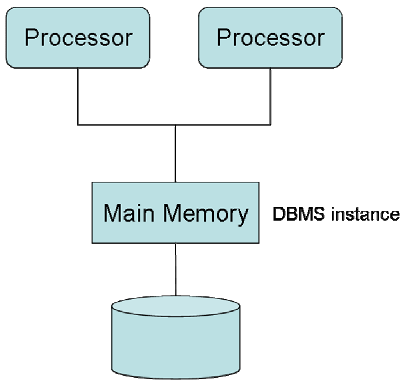
# 4 Parallel Database Systems

Same as in DDBS, in parallel database systems (PDBS), the logical database can also be distributed to multiple nodes by fragmenting and replicating. Its difference to DDBS is primarily that the computers are located close to each other (e.g. in the same building). Therefore they can be interconnected through fast links. This adjacency and good connectivity can be used to process transactions or operations in parallel to improve performance and throughput of the system. From an HA point of view, replication in PDBS causes lower overhead than in DDBS, thus a highly up-to-date replica or standby becomes possible. Similar to distributed DBMS, parallel DBMS should also provide the full functionality of a centralized DBMS while maintaining distribution transparency. In this chapter, we will introduce the four typical architectures of PDBS: *shared everything, shared nothing, shared disk,* and *shared data.*

## 4.1 Shared Everything

The architecture of a multi-computer database system, in which computers are tightly coupled, is called *shared everything* (also called *shared memory*). In other words, we can call a DBMS running on a multiprocessor shared everything. Figure 11 shows a DBMS running on an SMP with two identical processors.

**Fig. 11.** Shared Everything

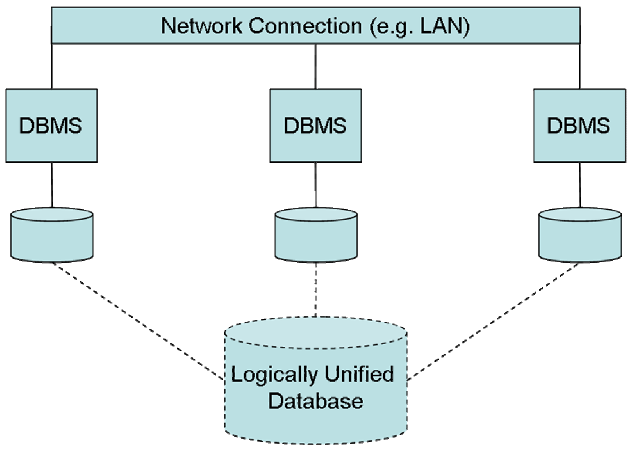


In order to make use of the computing capability of multiple processors, the DBMS should be able to do multiprocessing with the support of the operating system. For transactional and database applications, the operating systems nowadays support multiprocessing with up to 16 CPUs [2]. From a DBMS point of view, a strong advantage of shared everything is its simplicity, because issues such as load balancing can be efficiently handled using the shared main memory and these issues are normally handled by operating systems. However, it has the inherent disadvantages of tight coupling: limited extensibility and lack of error isolation. Therefore shared everything is not very popular as the stand-alone architecture of commercial DBS products today. *Instead, it is often used as the node architecture*, in the context of other architectures such as shared nothing and shared disk (see below).

## 4.2 Shared Nothing

In a *shared-nothing* architecture, the database is processed by a couple of loosely coupled independent computers. An instance of DBMS runs on each computer. The external storage (i.e. disk) is partitioned so that each DBMS instance has direct access only to its own partition. The logically unified database is distributed to the nodes by fragmenting and replicating (see Figure 12). Each computer in a shared-nothing architecture can be a multiprocessor. Thus the DBMS instance running on this computer (or better called a node) can be viewed as a "shared everything".

**Fig. 12.** Shared Nothing

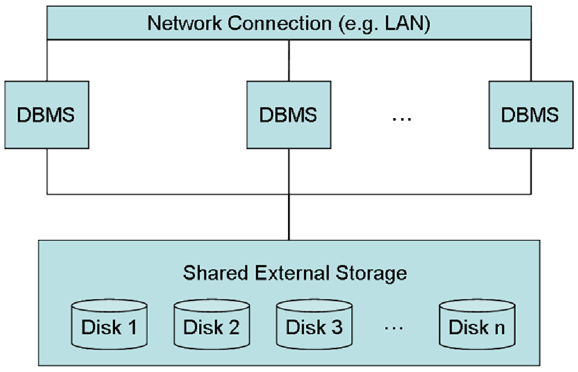


A shared-nothing parallel DBS is very similar to a distributed DBS, except that the nodes are interconnected through fast links, e.g. high-speed LAN. Sometimes distributed DBSs are also described as shared nothing. With a shared-nothing architecture, it is possible to achieve high availability by replicating data across nodes (protection against node failure) or sites (protection against disaster, in a geographically distributed case). It is also possible to improve performance, if queries can always be processed close to the nodes holding the relevant fragments or replicas. However, this requires optimal allocation of fragments and replicas combined with efficient distributed query plans. Shared nothing suffers from further high complexities such as distributed concurrency control, distributed commit protocol, and replica updating algorithm, etc. We have discussed them in chapter 3 for distributed DBS, but they also apply to shared-nothing PDBS in most cases.

## 4.3 Shared Disk

As the name already indicated, in a *shared-disk* architecture, the external storage is shared by a group of loosely coupled computers working together. Each computer runs an DBMS instance. All of them have direct access to the shared external storage. Same as in shared nothing, each computer can itself again be a multiprocessor. Figure 13 shows the basic structure of a shared-disk database system.

**Fig. 13.** Shared Disk



Since each node has direct access to all the data, all the operations of a transaction are executed at one node. There is no need for distributed query plans and distributed commitment protocols. However, concurrency control will be necessary to ensure the consistency of the database, because now a data object can be accessed simultaneously by multiple nodes. Furthermore, since the same database page can be buffered and updated by multiple nodes in their own main memory, there must be some mechanism to keep the buffer coherent. Concurrency control and buffer coherency control causes considerable overhead, because the inter-node communications required (e.g. message broadcasting) are very expensive.

In spite of these problems, when compared with a shared-nothing architecture, shared disk still exhibits great advantages from the perspective of high availability and scalability. In shared disk, a node failure has no impact on the data availability, because all the data are still accessible to the surviving nodes. In shared nothing, if a node fails, its data partition can not easily be accessed by other nodes, because each node has direct access only to its own partition. In shared disk, to keep up with increased workload, new nodes can be easily added to the system and take a part in the query processing. In shared nothing, however, a new node can not take an active part in the query processing immediately, until it has direct access to some data. For direct access, these data must be located in its own partition. This results in a reallocation of possibly the complete database, which is very expensive and may cause the database unavailable for a long time.

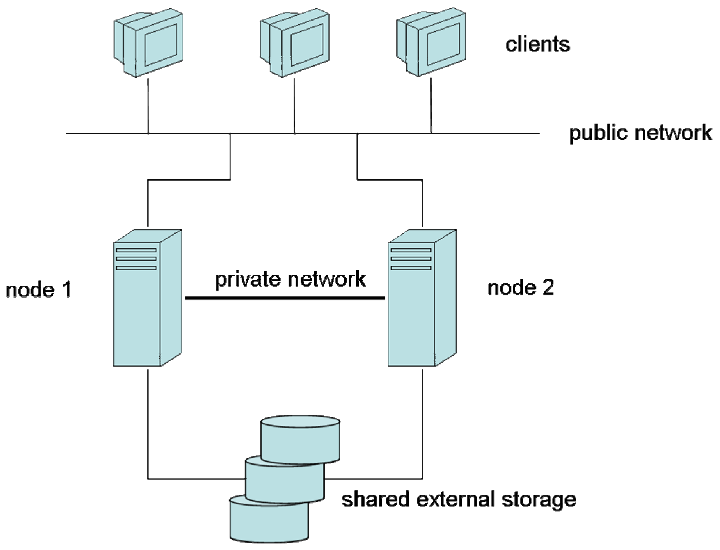
### 4.4 Shared Data

*Shared data* is intended to eliminate the disadvantages of shared disk while keeping the advantages. It can be viewed as a variant of shared disk, where the nodes all have direct access to the shared external storage. Its difference to pure shared disk is that the nodes are closely coupled. That is, besides their private main memory, to which they have exclusive and direct access, a large global main memory is shared between them. The global shared memory can be used for inter-node data exchange, thus reducing the communication overhead, which we discussed with shared disk. Tasks such as global concurrency control and buffer coherency control would be expensive in case of pure shared disk due to a possibly large number of messages involved. We have already introduced an implementation of shared data in section 2.1.4 with the Parallel Sysplex architecture as example.

# 5 Clustering

Clustering is the interconnecting of a group of computers so that they work together closely and in many respects it can be viewed as if it were a single computer (single system image). Computers in a cluster are commonly, but not always, interconnected through fast local-area networks. According to how the computers are interconnected, clusters can be classified into the categories of *tight coupling, loose coupling*, and *close coupling* (see section 2.1). The general objectives of clustering are achieving higher availability and performance than a single computer has. From this perspective, clusters can be classified into *high-availability clusters* and *load balancing clusters*. From an application point of view, the architecture types introduced in last chapter, *shared nothing, shared disk,* and *shared data*, apply not only to PDBS. Instead, they also apply to clustered systems. The following figure is a shared-disk cluster with two nodes.

**Fig. 14.** Cluster with Two Nodes and Private Network



### 5.1 High-Availability Cluster and Load Balancing Cluster

*High-availability cluster* is also called "*fail-over*" cluster. In the simplest case, fail-over means there is a production server (also called primary) and a backup server in the cluster. The backup has normally the same configuration as the primary, so that, if the primary fails, the backup can take over its processing with little or no impact to users or applications. Before the primary fails, the backup server can be already running and in sync with the primary so that it is able to take over very quickly, e.g. in sub-second time. This is called a *hot standby*. In some configurations, the standby is to be initialized before it can take over the workload. For example, the client connections need to be rebuilt or crash recovery needs to be done by the standby DBMS instance, etc. This is called a *warm standby*. In an HA cluster, the backup is idle most of the time, i.e. it doesn't take any workload until the primary fails. This is because, if the standby is also processing workloads, it may not have enough resources available to take over the workload of the primary.

HA clusters provide effective protection against unplanned downtime, because fail-over happens automatically and fast. An HA cluster can also be used to reduce planned downtime, e.g. an upgrade of the operating system. In this case, the primary is taken offline manually by administrator and the backup is assigned the primary role, so that the services provided by the old primary are continued by the new primary while the old primary is being upgraded. This is called a "*switch-over*".

A *Load balancing cluster* has the objective of improving performance and throughput. Therefore all the servers in a cluster are actively processing the workload. This is also its major difference to an HA cluster. It is also possible that an HA cluster is combined with an LB cluster, e.g. a few active nodes processing the workload and one idle node ready to take over if one of the active nodes fails.

## 5.2 Private Network

To fail over automatically, the system needs to detect a node failure quickly. This is normally realized with the so-called “heartbeat” messages, which are status information sent by the nodes to each other at regular intervals. However, this does not deal with the network partition problem resulting from connection failures. In our simplest case above, if a connection failure happens, both the primary and standby are running, but both of them may think that its partner is dead due to the loss of heartbeats from the partner. Then the standby would try to take over, resulting in two primaries. For a database, it would end in a chaos. This is the so-called “split brain” situation.

The Microsoft Cluster Server (MSCS) [17] uses a “quorum resource” to deal with the split brain problem. The quorum resource is saved on shared disk so that both nodes have access to it, but only one node can own it. So in the case of a connection failure between the nodes, the node owning the quorum resource can continue to operate in the cluster. This approach implies the shared-disk architecture.

The communication links between primary and standby are also important to keep the standby up-to-date or in sync with the primary. For these reasons, the connection between primary and backup is often recommended to be redundant, fast, and reliable. The network for this kind of purpose is called *private network* or *dedicated network*, its counterpart is the so-called *public network*, which is the network where the client connections come from.

## 5.3 Single System Image

A *single system image* (SSI) is the property of a system that hides the distributed (and possibly heterogeneous) nature of the available resources and presents them to users and applications as a single unified computing resource [14]. SSI describes the interface provided by the cluster to users and applications. *Single entry point* [16] is one aspect of SSI. For example, there might be several DBMS instances running on multiple physical nodes in a cluster, but an application only needs to connect to a virtual host (e.g. using one IP address). This is of great importance for high availability, because a possible fail-over behind the virtual host is then transparent for the application, and no downtime can be observed by the application. SSI can be achieved at different levels, ranging from hardware, operating system, to application. The IBM modular multi-book design can be viewed as a hardware approach for SSI, where the books (SMPs) are interconnected via L2 cache to form a logically single large SMP (see section 2.1.4).

### 5.3.1 SSI at OS Level

The commercial releases of cluster operating systems such as Microsoft's Windows NT/2000/2003 and IBM's AIX provide SSI at the operating system level. Under Windows NT, the cluster appears to network clients as a single server providing *services*. These services are provided by *resources*, which are any entity on a physical server that can be managed by clustering software and that actually provides a service to clients. For example, a public file share, a Web server, and a database application can all be managed as resources. Network clients connect to cluster resources the same way they connect to those resources on any physical server [17].

### 5.3.2 SSI at Application Level

The application-level SSI is the highest and, in a sense, most important abstraction, because this is what the end user sees. The Oracle RAC (Real Application Clusters) is an example of clustering with SSI at application level, if we look at the DBMS as a kind of server application. RAC is an option of Oracle Database providing cluster service for the Oracle Database. It detects failures of DBMS instances, or executes a fail-over or restart. Though cluster operating systems can detect node failures, but they may not know if a DBMS instance crashes or need to be restarted. Most client connections to the DBMS are made using the TCP/IP protocol. RAC floats the IP address of the failed nodes to a surviving node as part of the recovery procedure [18], ensuring that requests even to the failed node have a destination so that clients do not need to make the connection again.

### 5.3.3 SSI Middleware

SSI can also be achieved at middleware level. C-JDBC [19] is an open source middleware which allows clustering of heterogeneous DBSs. It turns a collection of possibly heterogeneous databases into a single virtual database. The heterogeneous databases are accessed by C-JDBC through their native JDBC driver. Client applications can access the single virtual database using C-JDBC driver, which implements the JDBC 2.0 specification. For high availability or load balancing, C-JDBC allows data to be replicated or partitioned to the table granularity. Because clustering of heterogeneous DBS is allowed, the C-JDBC implies a shared-nothing architecture. Furthermore, the ROWA strategy is implemented for replicated data. Thus performance may suffer when the number of nodes increases. Above all, C-JDBC is a very flexible approach: even a C-JDBC virtual database can again be a node in a cluster, thus allowing cascaded clusters.

# 6 HA in Commercial DBS Products

For the marketing campaigns of major DBMS vendors such as IBM, Oracle and Microsoft, etc., HA is one of the focuses. Therefore great efforts have been made to enhance the HA aspects of their products. Numerous HA technologies and features are supported by commercial products like DB2, Oracle Database and SQL Server. Properly understanding and applying them are essential to achieving HA.

As introduced in chapter 1, the primary causes of unplanned and planned downtime are computer failures, data failures, data changes, and system changes. In the following sections, we will in turn look at theses causes of downtime and how they are addressed by the commercial DBS products today. In doing so, we mostly use the HA features and technologies in Oracle Database 10g as example [22].

## 6.1 Unplanned Downtime

### 6.1.1 Computer Failure

A computer failure can be caused by server hardware failure and server crash. The direct results of computer failure are loss of data in main memory and database service interruption. These types of failures are addressed by database crash recovery and clustering technology. Using log files for database crash recovery is a common practice. Since major commercial DBS products implement a no-force strategy for end-of-transaction handling, the recovery process can potentially take a long time. Checkpointing is an approach which limits the time of recovery [5]. Oracle Database self-tunes the frequency of checkpointing to limit crash recovery time and thus makes crash recovery time predictable. Clustering is supported by Oracle Database with the RAC (real application clusters, see last chapter) option, which remedies the service interruption caused by a computer failure by allowing the crashed services to fail over to another instance running on a surviving computer.

### 6.1.2 Data Failure

A data failure is the loss of critical data. Data failure can be caused by storage failure, human error, or site failure.

*Storage Failure*
Storage failure means disk crash or disk defect. The most common protection against storage failure is to use redundant disk arrays and backups. Oracle Database provides an Automatic Storage Management (ASM) feature, which can do mirroring at the database file level instead of having to mirror the entire disk. For applications which require continuous availability, the major commercial DBS products support online backup. Oracle provides a tool called RMAN (Recovery Manager) that manages the backup, restore, and recovery process for the Oracle Database. In IBM DB2 UDB [20], an online backup can be started, for example, using the command:

```
db2 backup db sample online to /dev3/backup
```

*Human Error*
Many research results show human error as the single largest cause of downtime. Examples of human errors are: deletion of critical data or an incorrect WHERE clause in an UPDATE or DELETE statement. Against human errors commercial DBS products have provided means for error correction, such as the Flashback technology of Oracle. Flashback provides an SQL interface to analyze and repair human errors. It supports recovery at various levels ranging from row to database. The DBMS automatically keeps the necessary information to reconstruct data for a configurable time into the past.

For example, the following statement is used to retrieve rows from the employee table as of 2 o'clock in the afternoon of some day in the past.

```
SELECT * FROM employee
AS OF TIMESTAMP TO_TIMESTAMP('31-DEC-05 02:00:00 PM')
WHERE …
```

Dropped tables are first kept in a Recycle Bin until the user decides to permanently remove them or there is a space pressure. Tables kept in the Recycle Bin can be recovered. For example, the following statement recovers the employee table and all its dependent objects.

```
FLASHBACK TABLE employee TO BEFORE DROP;
```

Normally, recovery of a database from backups takes a lot of time. It may causes hours or even days of downtime. Oracle uses the so-called flashback logs to capture old versions of changed blocks. When recovery needs to be performed, the logs can be used to restore the database very quickly. For example, a database can be recovered to an earlier point in time by using statements like:

```
FLASHBACK DATABASE
TO TIMESTAMP TO_TIMESTAMP('31-DEC-05 02:00:00 PM')
```

The Flashback technology may consume considerable disk storage. But the cost of disk storage today is so low (and still being reduced), that this consumption should be acceptable for the goal of faster recovery.

*Site Failure*

Site Failure means that some catastrophic events preventing a site from functioning for an extended period of time. Examples are natural disasters, regional power failure and communication outages. Keeping backups off-site so that after a site failure the off-site backups can be used for recovery is the simplest form of protection against site failures. However this kind of recovery is time-consuming and the backups are not up-to-date. A better solution is to have a standby system at another site, which is being kept up-to-date by applying the same changes made to the primary database so that in case of site failure the standby can take over the production role very quickly and continue the service. The redo logs of the primary database are sent to the standby. This is the solution we discussed in section 3.4 for distributed DBS.

Oracle Data Guard is an implementation of this solution. Data Guard is the software that maintains the standby database. In addition, it manages recovery, switch-over, and fail-over between sites. Data Guard provides three protection modes: maximum protection, maximum availability, and maximum performance, which can be set using the SQL statement:

```
ALTER DATABASE SET STANDBY DATABASE TO MAXIMIZE
{PROTECTION | AVAILABILITY | PERFORMANCE}
```

The three protection modes correspond to very-safe, 2-safe, and 1-safe respectively which we discussed in section 3.4. Data Guard supports two kinds of standby databases: physical standby and logical standby databases, both of them can be used to take over processing if the primary database is unavailable, e.g. due to a site failure. The physical standby database is physically identical to the primary. It is maintained like a recovery process by continuously applying the redo data shipped from the primary database. The logical standby database is logically the same as the primary. It can have a different physical structure than the primary, e.g. it can have additional indexes and materialized views that do not exist in the primary. Data Guard maintains the logical standby by transforming the archive logs of the primary into SQL transactions and applying them to the standby. To share the load of the primary, the logical standby database can be used for the processing of read-only transactions.

## 6.2 Planned Downtime

### 6.2.1 Data Changes

Data changes refer to the reorganization of schema and data. Planned down time can be reduced if most of these changes can be done online. For example, the Oracle Database allows the relocation and defragmentation of tables to be done online.

### 6.2.2 System Changes

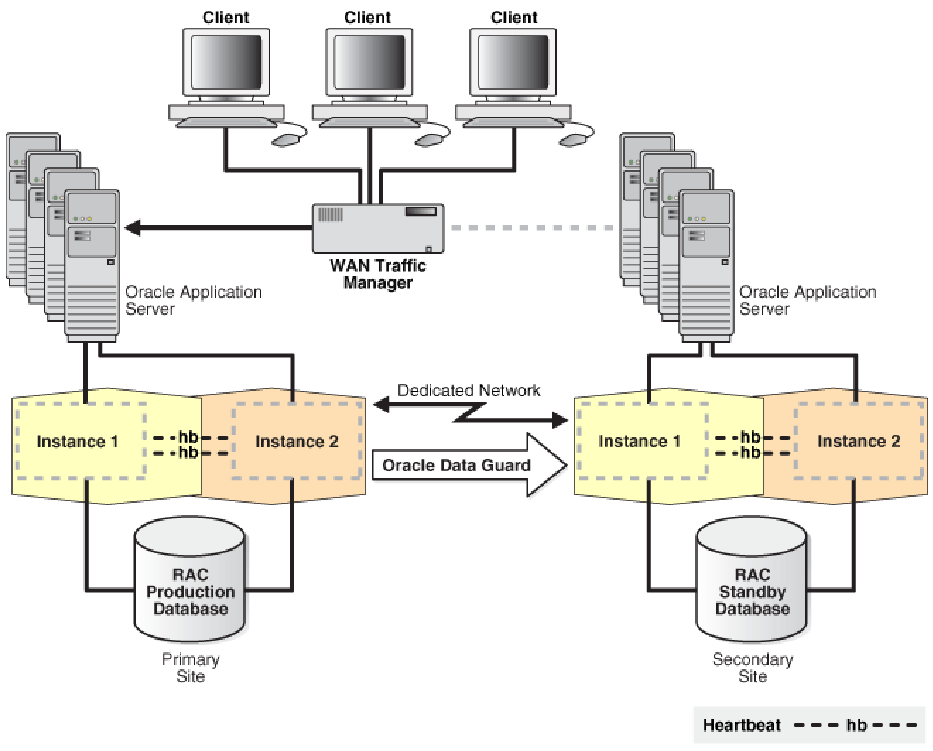
System changes primarily refer to maintenance works on the hardware of software like adding new hardware, installation of service pack of the operating system, etc. This kind of changes normally requires the system to be taken offline. They are best be remedied by the clustering technology: the maintenance work can be done on one node at a time, without taking the whole system offline. Major commercial DBS products such as DB2, Oracle Database, and SQL Server all support clustering. They either rely on the cluster-capable operating systems, examples are DB2 on AIX and DB2 on Windows, or supply the cluster ware as an option of their products, an example is the Oracle Database's RAC option which we introduced in section 5.3.2.

## 6.3 Operational Practice

Having a product with numerous HA features does not equal HA. HA can only be achieved by properly applying these technologies. However, this is not so easy for users. It is particularly difficult to design an HA architecture for the complete information system of an enterprise, with consideration of both performance and availability, and under proper application of various HA features and technologies. So the vendors collect sound HA architectures and good operational practice, test and validate them, then recommend them to users. These are of course also used as marketing instruments.

The so-called Oracle MAA (Maximum Availability Architecture) [21] is a collection of such a kind of HA architectures and operational practice. The key technologies in Oracle MAA are RAC and Data Guard, which provide protection against node failure and site failure respectively. For example, Figure 11 represents a configuration of the Oracle MAA:

**Fig. 15.** Oracle Maximum Availability Architecture



This architecture involves two sites: the primary and secondary. The primary site contains multiple application servers and a production database using Oracle Real Application Clusters (RAC) to protect from host and instance failures. The secondary site has an identical configuration. The database at the secondary site is kept synchronized with the primary database by Oracle Data Guard. Clients are initially connected to the primary site. If the primary site fails, Data Guard fails over the production role to the standby database, and the clients can be directed to this new primary database at the secondary site, thereby keeping the service available.

## 7 Conclusion

In this article, we have first introduced hardware platforms of highly available DBS using IBM mainframe products and virtual machine technologies as examples. Then we talked about DDBS and PDBS. These are theoretical fundamentals for modern HA technologies. Research done for distributed DBS covers replication and disaster recovery. Architectures and techniques in parallel DBS find similarities in the clustering technology. With clustering, the primary objective is fast fail-over and load balancing. Shared-disk architectures have inherent advantages for these objectives and bring lower complexity to the implementation. Therefore it is the most popular architecture for clusters at the time.

Since faults and errors in software and hardware components are a matter of fact, using redundancy to eliminate single points of failure is the intuitive and effective approach for high availability. Almost all HA technologies employ redundancy. Clusters can be viewed as a kind of *process redundancy*[4]. Even the external storage is shared (shared-disk architecture), the network connections between nodes are redundant, and the access paths to the storage are also redundant. In addition, *data redundancy*, i.e. redundancy of the external storage itself, can be implemented at the hardware level, e.g. by using RAID. Backups and remote standby site are also forms of data redundancy.

Highly available DBS represents a broad topic. From research to practice, there are many aspects that can not be included here. Or to be studied yet, for example, it may be a challenge to prove the availability of a system that is said to have achieved eight nines. There is also a promising market for HA DBS, because downtime is highly expensive.

## 8 Reference

[1] Erhard Rahm: Mehrrechner-Datenbanksysteme. Addison-Wesley, 1994

[2] Joachim von Buttlar, Wilhelm G. Spruth: Virtuelle Maschinen: zSeries- und S/390-Partitionierung. Informatik - Forschung und Entwicklung Juli 2004, Band 19, Seiten 41-54, Springer-Verlag

[3] Wihelm G. Spruth, Erhard Rahm: Sysplex-Cluster-Technologien für Hochleistungs-Datenbanken. Datenbank-Spektrum 3/2002: 16-26

[4] Theo Härder: Implementierung von Datenbanksystemen. Hanser Verlag 1978: 293-310

[5] Theo Härder, Erhard Rahm: Datenbanksysteme, Konzepte und Techniken der Implementierung. 2. Auflage, Springer-Verlag 2001: 471-477

[6] Sam Drake, Wei Hu, Dale M. McInnis, Martin Sköld, Alok Srivastava, Lars Thalmann, Matti Tikkanen, Øystein Torbjørnsen, Antoni Wolski: Architecture of Highly Available Databases. ISAS 2004: 1-16

[7] M. Tamer Özsu, Patrick Valduriez: Distributed and Parallel Database Systems. The Computer Science and Engineering Handbook 1997: 1093-1111

[4] For more details on process redundancy and data redundancy, see [6].

[8] Algirdas Avizienis, Jean-Claude Laprie, Brian Randell, Carl E. Landwehr: Basic Concepts and Taxonomy of Dependable and Secure Computing. IEEE Trans. Dependable Sec. Comput. 1(1): 11-33 (2004)

[9] Jim Gray: What Next? A Few Remaining Problems in Information Technlogy, SIGMOD Conference 1999, ACM Turing Award Lecture, Video. ACM SIGMOD Digital Symposium Collection 2(2): (2000)

[10] D. Brock: A Recommendation for High-Availability Options in TPC Benchmarks. http://www.tpc.org/information/other/articles/ha.asp

[11] Atul Adya, Robert Gruber, Barbara Liskov, Umesh Maheshwari: Efficient Optimistic Concurrency Control Using Loosely Synchronized Clocks. SIGMOD Conference 1995: 23-34

[12] Jim Gray, Andreas Reuter: Transaction Processing: Concepts and Techniques. Morgan Kaufman 1993, ISBN 1-55860-190-2

[13] Franck Injey et. al.: IBM System z9 109 Technical Guide. ibm.com/redbooks

[14] Chee Shin Yeo et. al.: Cluster Computing: High-Performance, High-Availability, and High-Throughput Processing on a Network of Computers. Handbook of Innovative Computing, Springer Verlag 2005

[15] Gopal Krishnan: IBM Mainframe Database Overview and Evolution of DB2 as Web Enabled Scalable Server. Datenbank-Spektrum 3/2002: 6-14

[16] R. Buyya et. al.: Single System Image. The International Journal of High Performance Computing Applications, Volume 15, No. 2, Summer 2001, pp. 124-135

[17] Microsoft: Windows NT Server Products Documentation, Chapter 1 - MS Cluster Server Concepts. http://www.microsoft.com/technet/archive/winntas/proddocs/mscsadm1.mspx

[18] Oracle: Building Highly Available Database Servers Using Oracle Real Application Clusters, Oracle White Paper

[19] Emmanuel Cecchet, Julie Marguerite, Willy Zwaenepoel: C-JDBC: Flexible Database Clustering Middleware. USENIX Annual Technical Conference, FREENIX Track 2004: 9-18

[20] IBM: DB2 Information Center. http://publib.boulder.ibm.com/infocenter/db2luw/v8//index.jsp

[21] Oracle: Oracle Maximum Availability Architecture Overview http://www.oracle.com/technology/deploy/availability/htdocs/maa.htm

[22] Oracle: Oracle Database 10g Release 2 High Availability, Oracle White Paper