Distributed Databases.

Defn: A distributed database is a collection of data that logically belongs to the same system but is physically located at multiple sites.

Advantages

1) Data is often naturally divided into local vs global data. A DDB (Distributed Data Base) allows data to be stored where it is used.
2) Increased reliability (the percentage of time the system is up) and availability (probability that a system is available over some time period). Reliability and availability can be further increased by allowing the duplication of data...
3) Local control over database and software
4) Improved query performance. data may not have to be transmitted from node to node.

Disadvantages:

1) Duplication of data
2) Transaction processing is MUCH harder.

DDBMS’ require much additional functionality:
1. Ability to access remote sites & to transmit data among remote sites.
2. The data dictionary must be able to keep track of data distribution and replication.
3. More complex query execution and optimizing strategies
4. Replicated data must be kept consistent.
5. Networks introduce new types of failures so recovery strategies must be more complex. has a node failed or has the data transmission link failed?
6. The Tx manager must be able to handle multi-node Tx’s.

Terminology:

Data transparency...goal of a DDB...all users can write queries against the DB as if there querying a local DB...i.e. they don’t need to know where the data is (Ingres/Star)

Data fragmentation. A data fragment is a logical unit of data storage.

Horizontal fragment is a subset of the tuples of a table...as if some restriction had been imposed. Horizontal fragments can be reunited by union.
Vertical fragment a subset of the attributes as if a projection condition had been imposed. Note that a vertical fragment must be accompanied by a primary key attribute since they are reunited by a join.

A set of horizontal fragments whose union is the original table is called a complete horizontal fragmentation. If no tuple appears more than one fragment, we say that the fragments are disjoint.
A set of vertical fragments, \( v_1, v_2, \ldots, v_n \) is called complete if

1. \( \text{UNION} \ s\text{ch}(v_i) = s\text{ch}(r) \)
2. \( s\text{ch}(v_i) \cap s\text{ch}(v_j) = \text{PK}(r) \), for any \( i \neq j \), where \( \text{PK}(r) \) is the primary key of \( r \)

A fragmentation scheme is called mixed if each fragment is determined by a select-project condition.

Defn: A fragmentation scheme for a relation \( r \) is a set of fragment definitions such that

1. All attributes of \( r \) are someplace
2. \( r \) can be reconstructed by the correct application of union and join and possibly outer join.

Defn. An allocation scheme is an allocation of various fragments to various physical sites.

Defn. If a fragment is allocated to more than one site, it is said to be replicated.

Replication extremes:

- Fully replicated...all data is present at all sites.
  - Advantage: Greatly improved query performance
  - Disadvantage: Tx processing is VERY DIFFICULT.
- No replication:
  - Advantage: Concurrency and recovery are no more difficult than a single site database
  - Disadvantage: Query processing may be VERY slow since a great deal of data may have to be transmitted.

Anything between the two extremes is called a replication schema.
Distributed query processing.

The major cost factor is the time required to transmit data over a network.

Example: (Elmasri & Navathe). Table EMP has 10,000 records of 100 bytes each stored at site 1. Table DEPT has 100 records each of 35 bytes stored at site 2. DEPT.mgrssn is 9 bytes, DEPT.dname is 10 bytes, EMP.lname is 20 bytes and EMP.fname is 10 bytes. The following query is issued at site 3 (site that is different from all the sites that contain files involved in the query):

```sql
Select fname, lname, dname
From EMP, DEPT
Where DEPT.mgrssn = EMP.ssn
```

The result of the query is 100 rows with 40 bytes each.

There are 3 possible strategies:

1. Send EMP & DEPT to site 3 & process the query there.
2. Send EMP to site 2, execute the query there and send the results to site 3.
3. Send DEPT to site 1, execute the query there and send the result to site 3.

Calculations:
EMP totals 1,000,000 bytes
DEPT totals 3,500 bytes,
Result totals 4000 bytes.
Cost of 1 is 1,003,500 bytes
Cost of 2 is 1,004,000
Cost of 3 is 7,500

If we assume that the query was issued at site 2, (site containing one of the files involved in the query), we have two simple strategies:

1. Send EMP to site 2, execute the query and present the result to the user at site 2
2. Send DEPT to site 1, execute the query at site 1 and send the result back to site 2.

Calculations:
Cost of 1: 1,000,000 bytes
Cost of 2: 7,500 bytes

A more complex strategy, which sometimes works better that these simple strategies, uses an operation called semi-join.

The idea is to reduce the number of tuples in a relation before transferring it to another site.
Definition: A semi-join operation $R \bowtie_{A=B} S$, where $A$ and $B$ are domain-compatible attributes of $R$ and $S$, respectively, is the join of the projection of $S$ on $B$, i.e. $\Pi_B(S)$, with $R$. Therefore, the semi-join produces the same result as the $\Pi_R(R \bowtie_{A=B} S)$. Note that the semi-join is not commutative, i.e. $R \bowtie_{A=B} S \neq S \bowtie_{A=B} R$.

Consider the previous example, when the query was issued at site 2. There is a new strategy involving the semi-join:

1. Send DEPT.mgrssn to site 1. Perform semi-join at site 1. Send this back to site 2 and perform the final query at site 2.

Calculations:
Cost of 1:
DEPT.mgrssn contains 900 bytes.
Semi-join produces 100 rows of 39 bytes each i.e. 3900 bytes. So the total cost is 4800 bytes.

Concurrency Control and Recovery in DDBs causes some special problems that do not arise in a centralized DBMS.

1) Dealing with multiple copies of the data items.
   Concurrency: Consistency must be maintained among all the copies.
   Recovery: This process must make a recovered fragment consistent with other copies
2) Failure of individual sites.
   Concurrency: A local database may continue processing when another site fails.
   Recovery: When the failed site is brought back, it must be brought up to date.
3) Failure of communication links.
   Concurrency: Must be able to deal with network failures. It could happen that the network is partitioned into two or more partitions, each one able to communicate with the nodes inside, but not to the nodes in another partition.
   Recovery: When the connection is established, all nodes must have consistent copies
4) Distributed Commit.
   Committing a Tx may involve multiple sites (and local databases)
5) Distributed deadlock.
   Now the deadlock may occur not only among transaction working on the same site, but also deadlock can arise among different sites, so the DDBS system must deal with this situation.

Some solutions.

Let’s deal first with the problems of Distributed Concurrency Control.

First Try. Use a “distinguished” copy of each data item. All locks and lock requests are sent to the site having that copy. There are different ways to choose the distinguished copies.

- Primary site technique: One site is designated for all locking requests...even if you only want to access a local database item, you must still obtain a lock from the primary site
Advantages: relatively simple, two phase locking guarantees serializability. It works like an extension of a centralized DBMS.
Disadvantages: overloads primary site. If it fails, the whole system dies. This limits availability and reliability. On the other hand only the locks have to be passed around. The actual data can be accessed at any site where the data is kept. When a write occur, and a write-lock is released, the DDBMS is responsible for updating all the copies containing information involved in the update.

-Primary site with backup. Two copies of all locking information are maintained at different sites. A Tx must obtain locks at both sites before it can continue.
Advantages: Tx’s don’t have to be aborted if the primary site fails. A new backup site must be chosen.
Disadvantages: Slows processing, because all locks tables must be updated on both sides before the transaction may proceed. Overload of both primary and backup copies may also occur.

-Primary copy. For every data item, one site is designated as containing the “primary” copy of that item. This site handles lock requests for that item.
Advantages: If one site fails, only Tx’s accessing data whose locks are at that site are affected...other Tx’s can continue.
Disadvantages: Processing is more complex, since a Tx has to know which site to request a lock from.
Variations
Use backup sites for each primary copy. It enhances availability and reliability, but it slows processing.
Token passing...The primary copy site for any data item is said to hold the token for that site. The token can be passed to another site, making it the primary site for a variety of reasons.

In all cases, the site that handles the locking for a data item is called its coordinating site. If a coordinating site fails:

Primary site or Primary copy techniques: all running Tx’s effected by the failure must be aborted and rolled back. A new primary site must be chosen and, perhaps, a new lock manager process started.
Sites with backup: Tx’s must suspend processing while control is transferred to the backup site and a new backup site is chosen, normally by the first backup site, and all lock info is duplicated at the new backup site. If both sites fail, a new coordinator site must be chosen from among the running sites.

In the case of choosing a new primary site, or in the last case when both sites fail, and new backup sites have to be selected, a process called election can be used to make this selection. If a site cannot communicate with the coordinating site, it may assume that it is down. It submits a request to become the new coordinating site. If the majority of the running sites accept the proposal, the requesting site declares itself and it becomes the coordinating site. There
is an algorithm that takes care of this election process. Once a new coordinating site has been elected, it may select the backup site, if it is required.

Another Try: Concurrency control based on voting.

There is no distinguished copy in this approach. Locks are maintained at all sites containing a copy of the data item.

Voting: When a Tx requests a lock, the request must be sent to all such sites that contain a copy of the item. Each copy contains its own locks for that item, and can grant or deny the request. If a majority of the sites grant the lock, the lock is assumed to be held and all sites are notified of the fact.

If a majority of sites do not grant the lock in a specified time period, the lock is assumed to be denied and all sites are notified of that fact. So we have a truly distributed concurrency control, because the responsibility of the decision lies on all sites involved with the data item.

The idea is that all sites must have identical lock values for any data item...the only reason some site would not reply that it has granted a lock is network traffic or it is too busy itself. If a failure occurs during voting, recovery is very complex. Network traffic can be increased but throughput can be very good.

**Distributed Recovery** from failure-: can be very hard.

ASIDE: it can even be very hard to determine if in fact a site has failed. It might be the network or the network software.

For instance, a site A sends a message to site B, and expects a response that it does not receive. We could attribute the failure to receive a response to the following different causes:

a) The site B never received the message, due to network failure.
b) The site B is down
c) The site B sent the response, but it never arrived to site A due to communication failure

In any case, additional information and additional messages must be used to assess the real situation.

Distributed Recovery using the two-phase commit protocol: A global Tx manager, or coordinator, must be used besides the local recovery managers and the information they keep. In this case all Tx’s must go through 2 phases:

In phase 1, the Tx manager sends a “prepare to commit” message is sent to all sites containing a replicated copy of the data. Each one of these sites receiving the message performs a force-write to all log records and then sends a “ready” to commit” message back to the manager.

If all sites reply correctly, the manager enters phase 2. If for some reason, some local DBMS cannot successfully do its part (maybe the disk containing the log has failed, the local
transaction cannot commit, it sends a “not OK” message to the global Tx manager. On the other hand, if after some fixed time period the manager receives no reply from some local site, it assumes “Not OK”

In Phase 2, the manager sends a “commit” message to all sites and each site writes a commit Tx record to the log and send a “committed” message back to the manager. If some site replies “Not OK” during Phase 1, the Tx must be aborted. This means that local sites that did reply correctly must use their local logs to roll back the Tx. If some site fails during phase 2, recovery is possible because all local log files contain the necessary redo information.

The net effect of the two-phase commit protocol is that either all participating databases commit the transaction, or none of them do. If any of the participating sites or the coordinator fails, it is possible to go back to a state where the transaction is either committed or rolled back.

EXERCISE: Try to figure out what happens if the Tx manager fails
   During Phase 1
   Between Phase 1 and phase 2
   During Phase 2

Failure during Phase 1: It requires the roll back of the transaction.
Failure between Phase 1 and phase 2: Think about it.
Failure during Phase 2: A successful transaction can recover and commit.

Client-Server Architecture and DDBMS

Full scale DDBMS, which support all the functionalities we have been mentioning, have not been developed yet. At this stage, the functionalities that are been developed for the DDBMS are always been done in the context of a client-server network architecture.

There are no rules in how to divide the DDBMS functionality between the client and the server.

One approach:

   SQL servers are provided to the clients, and these servers have the functionality of a centralized DMBS. The client must know the distribution of data among the various SQL servers. It also must have a way to break down its original query into small pieces corresponding to the local queries that must be sent to the various servers.

   So, we can summarize these actions in the following way:

   a) The client parses the global query of the user, and breaks it down into small queries that are sent to different servers.
   b) Each one of the involved servers processes the query it has received, and sends the results back to the client.
   c) The client combines the results received from the servers, and present the results to the user.
Here each one of the servers acts as a transaction server, and the client acts as an application processor. Note that the breaking of the global query at the client site can be done either by the user itself, or by a specialized module. The former approach is cheaper, but requires a lot of involvement and knowledge on the part of the user.

In a typical client-server DDBMS implementation, the software is divided into three levels:

1. Server software, which manages the server site in a local manner.
2. Client software, which takes care of the distribution part of the query, like accessing the information of where the data is distributed, sending the small queries to the corresponding sites, collecting the responses, etc.. It also must interface with the user.
3. Communication software, which provides the communication commands so the client can send its requests to the different sites. Even if this part of software should not be considered to belong to the DDBMS, it provides important functions for the DDBMS to work properly.

In this case, the client is in charge of creating a distributed query-execution plan, selecting the sites, sending the proper commands to perform local queries or to send data to other sites (server, or different clients). The client is also in charge of keeping the consistency of the different copies of the same data. Therefore it must employ some of the concurrency techniques that we saw previously, as well as global recovery techniques.

In some cases the distribution on the client is transparent to the user of the systems (they enjoy distribution transparency), while in other it must be performed by the users, requiring that they are aware of most of the details of data distribution.