Parallel, Distributed and Client Server Databases

Parallel Databases:

This is a database system running on a parallel computer.

Why parallel DBs? To make processing faster. Intuitively place different parts of a large table on different processors and perform queries in parallel.

3 kinds:

1. Shared Memory Parallel Databases

Several processors share RAM memory and also disks.

2. Shared Disk Parallel Databases

Each processor has its own memory, but they share the disks.

3. Shared Nothing Parallel Databases

Each processor has its own memory and its own disk(s). The processors communicate through a high-speed network.

Due to the increased speed of Local Area Networks (LANs) and Wide Area Networks (WANs) Shared Nothing parallel databases have become very similar to distributed databases. The main difference is in parallel computers the use of components that are all the same. Same hardware, same system and same software. And they are in one rack. Keep the distance between CPUs short, because long wires mean slow communication.

One of the main parallel processing initiative in the US is IBM's Blue Gene computer for the protein folding problem.

http://www-03.ibm.com/systems/deepcomputing/solutions/bluegene/

A protein initially exists as a linear "string" but it folds up very quickly when "let go". The forces between atoms result in a complicated shape, which is the "folded" protein.

Blue Gene was used to model how these proteins are folding up. A very hard problem!
(Tera = 10^{12}, FLOPS = Floating Point Operations per Second).

see: http://www.top500.org/sublist/results

1 RIKEN Advanced Institute for Computational Science (AICS)
Japan K computer, SPARC64 VIIIfx 2.0GHz, Tofu interconnect
Fujitsu 548352 8162 8773.63

2 National Supercomputing Center in Tianjin
China NUDT TH MPP, X5670 2.93Ghz 6C, NVIDIA GPU, FT-1000 8C
NUDT 186368 2566 4701

Speeds in Teraflops.

WATSON is also parallel.

See:

ORACLE supports some parallel execution.

select /*+ parallel(orders,4) */ count(*) from x;

BUT parallel processors (multi-core architecture) are now a big topic.

--------------

Distributed Databases (DDBs):

Distributed DBs have been studied extensively in theory. Existing systems (Oracle) have some features of DDBs, but no "complete" implementation exists.

M. Tamer Ozsu pioneered this field.

Why distributed databases?

Large organizations are distributed over the whole US and whole world and need to share data.

What if there is a big blackout?

Better reliability and availability. If the same data are stored at several places (see later) then queries can still be processed even if one computer is down.
(Like for Parallel): Better performance. Independent queries are performed by independent processors. Also, large tables might be broken into smaller tables (see later) and queries on smaller tables are faster.

Def: Reliability is the probability that the system is running at a certain point in time.

Def: Availability is the probability that the system is running continuously during a time interval.

Data Fragmentation:

Data is broken into pieces that can be stored at different locations. These pieces are called Fragments.

Horizontal Fragmentation:

A table is divided into groups of rows which can be placed at different sites. These are horizontal fragments.

If you have an employee master table of all employees, you might want to create a fragment with Los Angeles employees and store it in the LA office, and another fragment with all the Chicago employees and store it in the Chicago office. Same thing for New York.

That makes normal searches in each office faster. But you can still search everything from everywhere.

Complete Horizontal Fragmentation:

Each row of the original table exists in AT LEAST one fragment.

Disjoint Complete Horizontal Fragmentation:

Every row exists in ONLY one fragment. (Not necessarily a good idea...)

Vertical Fragmentation:

Tables are broken into groups of columns. In order to maintain the connection between rows, each fragment should have the primary key columns. Thus, vertical fragments are never disjoint.

Complete vertical fragmentation:
The union of all columns of all fragments gives the original table. The intersection of ANY pair of fragments gives the primary key columns. 
(Reminder: SET UNION, SET INTERSECTION)

Example:

Original Table:  {ID, Height, Weight, Age, Eye Color}
Fragment 1:  {ID, Height, Weight}
Fragment 2:  {ID, Age, Eye Color}

{ID, Height, Weight} UNION {ID, Age, Eye Color}  -->  {ID, Height, Weight, Age, Eye Color}

{ID, Height, Weight} INTERSECTION {ID, Age, Eye Color} --> {ID}

Mixed (hybrid) fragmentation is possible.

The information how data is fragmented is stored in the fragmentation schema.

The information at which site fragments are sitting is stored in the allocation schema.

Data Replication:

To achieve the improved availability and reliability, data must exist in several copies (at several sites). This is called replication. Thus, the same fragment might be stored at several sites.

Fully replicated DDB:

All data at every site. This is good in reliability and availability. However, tables are still large. Thus efficiency of access to local data is not helped.

Also, if data is frequently updated at several or all sites, keeping the replicated data synchronized is a nightmare. Thus, this is best if most data is read-only.

Nonredundant allocation:

No replication at all. This is bad in reliability but good in access to local data. (Every fragment at one place only.)

Different levels between these two extremes are possible, called partial replication.
Information about replicated data is kept in a replication schema.

Transparency:

Computer scientists like complicated mechanisms to be "transparent". For example, you never know where on your CD ROM or hard disk your files are sitting. You can call your editor (vi, WORD) on a file, no matter where it is sitting. This is a form of location transparency.

In DDBs:

Distribution Transparency:

User does not need to know where his data are. Two subcategories:

Location transparency: A command can be executed from any location and it will work.

Naming transparency: An object can be accessed with the same name from any location.

Fragmentation Transparency:
When the user poses a query he does not (need to) know that there are fragments at all.

Replication Transparency:
The user does not not need to know that there is replication.

Distributed Database Management System

Obviously, a DDBMS is much more complex than a DBMS, as it has to keep track of fragments, locations and replication. For example, it has to decide which replica to use that has the data it needs and decide on which processor (location) to execute a transaction.

Kinds of Distributed Databases:

Homegeneous system:
All nodes of the DDB run on the same hardware, use the same operating system (including same version) and same DBMS (including same version of query language).

Heterogenous system:

Everything else.
DDBs are also classified by their autonomy.

Centralized DDBMS: Looks like a system that is not distributed at all to the user. No local autonomy exists. There is a global schema.

Federated DBs: Each computer is independent (autonomous) but there is some global view or schema.

Multidatabase System: Autonomous systems with no global view or schema. But you can query one from the other.

ORDBMS Object-Relational Distributed Database Management System

Client-Server Architecture:

Client-Server architecture (in general, not just DB) grew naturally out of the fact that users of centralized main-frames started replacing their remote terminals by PCs and workstations. Thus, it was natural to do some work on "their own" processor instead of waiting for the mainframe to have time.

Modem picture:
See also ~/Teach/openhousepictures.ppt

Essence of Client Server Systems:
The client sends the call to the application (program) to the server and the application runs there and the client only receives the results.

Next, time consuming applications, such as printing, were placed on special servers to "get them out of the way": Print Servers, Email Servers, and much later Web Servers came out of this.

Thus a client is a user machine that provides local processing and user interface software. A server provides (what else) services such as printing and DATABASE ACCESS.

Approaches to DB C/S computing:

1) Interface programs and applications (database programs) run on the client. The server provides the data. Connection is done by ODBC (Open Database Connectivity). For Java: JDBC (Java Database Connectivity).

2) [Used in OODBs] The server deals with data at the "raw page level" (data server). Client and Server are tightly coupled. The interpretation of data as objects and everything else happens at the client.
Note! One hardware unit (PC, Workstation) may at the same time run both client software and server software, although this is not common for databases. It's common for Web servers.

Three-tier Client Server computing:

Client runs mostly the user interface.
Application server runs the database programs.
Database server runs the DBMS and accesses the database.

(In the context of the Web, the middle tier is the Web server. The client runs the Web browser as interface. The Web server itself invokes a separate database server.)

Today people say that the application server runs the "business logic".

Today, database servers often send their result to application servers in XML. Application servers send their results to clients in HTML.

-------------

Grid Computing
is now superseded by Cloud Computing.

The idea of grid computing was that you should be able to get computing power out of the "wall jack" just as you get electrical power out of a "wall jack." You pay for it only when you use it. You don't need to buy major hardware for your office.

Commercially not very successful.

Cloud computing: Based on "virtualization". That is software that allows you to ship a "memory image" of a computation from one server to another server. (For server farms.)

Commercial success. E.g. VMware.

The effect is similar to what the grid tried to achieve: You have computing power on demand.

-------------

Oracle:

Oracle's client server architecture supports location transparency.

Data in Oracle can be replicated.
Basic Replication: Data is replicated for read-only access.

Advanced (symmetric) Replication: Oracle uses "snapshots". Periodically the system is running a Snapshot Refresh Process.

Heterogeneous systems:

Oracle Open Gateways allows access to non-Oracle databases too.

----------------------