Distributed Database Management System

Distributed architectures allows data to be stored in different places or machines: depending on independence level of nodes it is possible to identify several level of complexity. This kind of approach allows to improve performances, increase avaibility and reliability.

Distributed architectures use a client/server approach: clients manage user interface while servers manage the database: this is a simple and widespread (among users) metodh. Although, in order to provide service, distributed databases have to communicate each other: it means that servers have to exchange information. This approach introduce much complex in order to guarantee ACID properties.

Data can be replicated: this is done only to protect database from failures; a replica, infact, is a copy of a data placed in another node of the network and updates of replicas are managed by replication manager which is a simpler architecture than distributed architectures.

Among distributed architectures it is possible distinguish:

. parallel architecture: their purpose is to increase performances;

. data warehouses: perform OLAP (On Line Analitical Processing) instead OLTP (On Line Transaction Processing) because are servers specialized in decision support.

There are two relevant properties:

. portability;

. interoperability.

The first one is guaranteed by SQL standard and specify the capability of moving a program to a different system; the second one, instead, is the capability of different servers to cooperate to a given project.

Client/server architecture

It is possible distinguish among two types of client/server architectures:

. 2-tier;

. 3-tier.

2-tier are typical because there are clients and servers users that cooperate each other; 3-tier architectures introduce a middle layer: the application server. In this way the client is very simple (a browser), the server still

manages the database and the application layer interacts between them (a web server).

What concerned the SQL execution, it can be:

. compile & go;

. compile & store.

In the first approach:

. the query is sent to the server;

. then it is prepared by generating the correspondent plan;

. after that execution is performed;

. then the result is avaiable.

With the second approach:

. the query is sent to the server;

. then it is prepared by generating the correspondent plan and stored;

. execution of program can continue and after some time query is executed;

. then the result is avaiable.

Distributed Database System

In these systems transactions performed by client can access more than a DBMS server but it introduce much complexity in order to find avaiable DBMS. It is important notice that locally DBMS are autonomous so concurrent control have to be provided to avoid concurrency problems.

This approach has several advantages:

. functional since an appropriate localization of data can be provided;

. technological because data avaiability is increased and scalability enhanced.

Distributed Database Design

Since each part of the database is managed by a different DBMS data have to be partitioned: this improve performances because each DBMS work independently and concurrently.

A data fragment is a subset of the entire relation: it can be a subset of tuples or a subset of the schema or both. This classification is done since, actually, there are three methods to fragment:

. horizontal;

. vertical;

. mixed.

Horizontal fragmentation is a sort of σ operation: tables are divided per tuple and the schema is preserved. The initial table exsist only virtually: it can be obtained by the union of all fragments.

Vertical fragmentation, instead, is a sort of π operation: the schema of the table is divided and each fragment has the primary key as an attribute (concept of *overlapping* the primary key: it is not present in horizontal fragmentation). Infact, the initial table, can be obtained by performing join operation on the primary key.

Mixed method uses both approaches. Fragments must not violate the following properties:

. completeness: each information in a relation must be contained at least in a fragment;

. correctness: the entire information must be obtained by summing all fragments.

The distributed database design is based on data fragmentation where each fragment is usually stored in different file on different server (if it is possible). The node who receive a query perform itself an execution plan and sends it to other servers, but they can decide to use another one because each DBMS is independent. Due to that fact *transparency levels* are defined: they describe the level of knowledge of the data distribution; there are:

. fragmentation transparency;

. allocation transparency;

. language transparency.

Applications operates differently according to the transparency level assigned.

Interaction among client and server take place in this way: first client requests the execution of a transaction to a DBMS server who have to redistribute it to others. There are several ways which a DBMS can choose to communicate to the others, according to the complexity level:

. remote requests;

. remote transaction;

. distributed transactions;

. distributed requests.