Parallel and Distributed Databases

Reference: Ch 21 Ed. 2, Ramakrishnan

Motivations for parallel and distributed databases, data partitioning, client-server architecture, fragmentation, distributed query processing.

Introduction

Parallel and Distributed databases are being increasingly used to achieve,

- better performance
- increased availability of data
- access to data distributed at various locations

A parallel db enables the first of the above, and a ddb enables the last two predominantly.

Parallel Database Architecture

Parallel databases (Pdbs) use more than one CPU/disk to carry out evaluation of a query faster.

There are three possibilities of how processors, disks and memory are shared.

- Shared nothing: the individual cpu/disk/mem are interconnected.
- Shared memory: All processors share the same memory, all data can move between memory and any disk.
- Shared disk: every processor has its own memory, but can r/w from any disk.

Communication overhead

The shared disk and shared memory architectures have communication overhead.

As the number of cpu/disk increases, there is more and more bottleneck in the communication.

Beyond a certain limit, increasing the number of cpu reduces the performance!

Shared nothing systems provide linear speed-up and linear scale-up (read text 21.1 to understand what this means), the desirable behavior. The time per Trans. should remain the same when the number of cpus and number of Trans. increase in proportion.
Parallel Query Evaluation

The query tree can be evaluated in parallel. Each of the leaf node can be independently evaluated, if the corresponding relations are stored in separate cpu/disk.

Even a simple query

select * from employee where age < 2

can be evaluated in parallel, if the corresponding relation is partitioned.

How to partition the relations among the different cpu?

Data Partitioning

The data can be partitioned in a number of ways.

- Each tuple is allocated to one of the \( n \) processors, in round robin fashion.
- Each tuple is hashed, and allocated to the corresponding processor.
- Each cpu stores a particular range of tuples, such as \( 35 < age < 40 \) goes to processor 3.

Which partitioning is better?

Depends on the nature of queries.

Distributed Databases

- A distributed system involves multiple sites or nodes connected together via communications network. The sites may be in the same building or geographically far apart.
- Each site has its own CPU, terminals, DBMS, users, DBA and local autonomy.
- A user can access data stored locally as well as from any other node.
- Distributed databases are becoming increasingly common, because they allow information to be collected/maintained/stored/used at one or more sites.
- DDBMS is also an out growth of the newly emerged computing environment.
Benefits of DDBMS

- A uses accesses the local data most of the time, and may need access to data stored at several locations occasionally. The DDBMS is a closer match to distributed applications such as airline reservation systems, bank ATM’s etc.

- increased RELIABILITY (probability that the system is running) and AVAILABILITY (probability that the system is continuously available over a period of time). If a site fails, the others can be operating. When coupled with data replication we get better system functionality.

- the balance between data sharing and local control can be achieved. Data and software can be controlled locally while the data is shared by remote users.

- improved performance is achieved because the processing of transactions can proceed in parallel at different sites. Local transactions access local data which is small as compared to a central system storing all the data and handling all the processing.

Costs of DDBMS

In order to be able to obtain the advantages of DDBMS there are several problems to be solved and additional costs incurred.

- communication between sites for transmission of data and commands
- capability to keep track of, index and process queries accessing dispersed and replicated data
- maintain consistence of replicated data
- provide capability to deal with situations such as a site failure or communication link failure

Client-Server Architecture

This architecture has become popular in commercial DBMS due to the advent of new computing environments where a large number of PCs, file servers, printers and other devices are networked together.

The idea is similar to file servers and disk less work stations networked together.

To reduce the complexity of the DBMS software, it is divided into client and server levels. There are different ways of accomplishing this division.
One such division is as follows. The client is responsible for providing user interface, programming language interface and formulate the SQL queries. Such a client is called a front-end machine or application processor (AP).

The server has the database engine and provides SQL service. Such a server is called a back-end machine or a database processor (DP).

In addition there is communication software (may also include a distributed operating system) which provides primitives for transmission of data and commands between the clients and servers.

The following steps are involved in a client processing a SQL query.

- the original SQL query is decomposed into independent site queries and dispatched to appropriate server site. In order to accomplish this, the client may refer to data dictionary which has information on the distribution of data among the SQL servers.
- the server sites process the local queries and send the result back to the client.
- the client computes the result of the original query from the results of sub queries.

These functions can also be accomplished manually where the user breaks the query into sub-queries and assembles the results obtained.

Functions of a client

- provide distribution transparency to the users. This refers to the situation where the user need not know the details of the data distribution and can pose global queries as if the database were centralized.
- the DDBMS keeps track of details of data distribution in a catalog. Client uses this information to break down the given query into a number of sub queries (generate execution plan) to be dispatched to server sites.
- not all DDBMS provide distribution transparency, and in such cases the user need to specify the location of data. This is typically done by appending the site name to the relation or file.
Data fragmentation

- ensure that the replicated copies of data are consistent. It may use distributed concurrency control techniques to accomplish this.
- perform global recovery to ensure the atomicity of global transactions.

Consider the relation

\[
\text{student(name, dept, age, address, ...)}
\]

of a university database.

This relation can be fragmented so that students belonging to a particular department are stored in the corresponding department site.

How?

**Horizontal fragmentation:**
A horizontal fragment of a relation consists of those tuples of the relation which satisfy a condition.

The condition may be on one or more attributes of the relation.

For the COMPANY database in Fig. 6.6 of the text, the condition \( \text{DNO} = 5 \) gives a horizontal fragment of the EMPLOYEE relation.

The fragment consists of all the EMPLOYEE tuples working for the department 5.

The fragment can be located where the department is located.

Each fragment is obtained by the select operation of the relational algebra, \( \sigma_{C_i}(R) \), where \( C_i \) is the condition.

If we obtain \( n \) fragments of a relation \( R \) using the conditions \( C_1, C_2, \ldots, C_n \), and these \( n \) fragments include all the tuples of \( R \), then the set of fragments is called a **complete horizontal fragmentation**.

How do we construct the original \( R \) from the fragments?

If no tuple in \( R \) belongs to more than one fragment, the fragmentation is said to be **disjoint**.

Examples of disjoint and non-disjoint fragmentation?
vertical fragmentation:
A vertical fragment of a relation $R$ is specified by the project relational algebra operation $\pi_{Li}(R)$, where $Li$ is the projection list.

Every vertical fragment should include the primary key of the relation. Otherwise, it will not be possible to reconstruct the original relation from the vertical fragments.

Example:

A set of $n$ vertical fragments on projection lists $L_1, L_2, \ldots, L_n$ is said to be complete, if the UNION of the $n$ lists gives the set of all attributes of $R$ and the INTERSECTION of any two lists is the primary key of $R$.

For the EMPLOYEE relation, the lists $L_1 = \{SSN, NAME, BDATE, ADDRESS, SEX\}$ and $L_2 = \{SSN, SALARY, SUPERSSN, DNO\}$ will constitute a complete fragmentation.

The original relation can be obtained by applying OUTER JOIN operation to the fragments. (Note that OUTER JOIN is similar to JOIN but includes all the non matching tuples with the corresponding values made null.)

Outer Join

Why OUTER JOIN, why not JOIN?

Consider a relation

emp(idno, officeno, phno) and its two vertical fragments empoft(idno, officeno) and emphno(idno, phno).

Consider the sample data as follows.

<table>
<thead>
<tr>
<th>empoff</th>
<th>emphno</th>
</tr>
</thead>
<tbody>
<tr>
<td>1001</td>
<td>B5.22</td>
</tr>
<tr>
<td>2085</td>
<td>C4.10</td>
</tr>
<tr>
<td>2650</td>
<td>S1.14</td>
</tr>
<tr>
<td>3222</td>
<td>A2.10</td>
</tr>
</tbody>
</table>

The outer join of the above two relations is,

| 1001 | B5.22 | 3-2011 |
| 2085 | C4.10 | 5-3020 |
| 2650 | S1.14 | 3-3034 |
| 3222 | A2.10 |       |
| 4333 |       | 3-1010 |

The information about 3222 and 4333 will be lost if we performed regular join. The regular join will have only three tuples.

| 1001 | B5.22 | 3-2011 |
| 2085 | C4.10 | 5-3020 |
| 2650 | S1.14 | 3-3034 |
Mixed fragmentation: The two types of fragmentation can be mixed. This corresponds to applying SELECT-PROJECT combination to the relation $R, \pi_L(\sigma_C(R))$, to obtain the fragment.

A fragmentation schema of a database is a definition of a set of fragments such that,

- all attributes and tuples in the database are included
- the database can be reconstructed from the fragments by applying a sequence of OUTER JOIN and UNION operations

An allocation schema is a mapping that specifies at which site each of the

---

Data replication and allocation

If each fragment is stored at only one site we say there is no replication. If the whole database is stored at every site, it is said to be a fully replicated distributed database. Partial replication is between the two extremes, when some of the fragments are replicated.

Data allocation or data distribution refers to the process of assigning the individual fragments to sites. The replication schema describes the replication of the fragments.

Advantages

Data replication increases the availability of data. If one site is down, the data can be obtained from the replicated site.

Replication reduces the communication cost of the global queries.

Disadvantages:

The copies of data have to be kept consistent and hence updates will be expensive.

Replication makes concurrency control and recovery techniques more expensive.
When and how much to replicate?

*General statements:* If the transactions at a particular site frequently access certain data, then the corresponding set of fragments should be allocated to the site.

If most of the transactions are retrievals (with few updates) and high availability is required, the whole database can be replicated at every site.

If many updates are performed, it may be better to limit replication.

Finding the best possible allocation and replication is a very difficult problem.

Types of distributed database systems

If all access to a DDBMS is through a client, then there is no *local autonomy*.

If local transactions can access the server data, there is local autonomy.

In a *homogeneous DDBS* all servers use identical software, and all clients use identical software. Otherwise, it is called a *heterogeneous system*.

A *federated DDBMS or multidatabase system* is an hybrid between distributed and centralized systems.

Each site is autonomous with every server being an independent DBMS with local users, local transactions, local DBA.

Each server permits portions of its data to be accessed by a class of nonlocal users through export schema. A client in such a system is basically an interface to autonomous databases.

A *heterogeneous multidatabase system* may consist of databases with different data models. One server may be relational, while the other may be a network database. The clients need to use a common canonical language, and the queries translated to the corresponding language for each server.
Distributed transparency refers to degree to which details of fragmentation, replication and distribution are hidden from the users. If a user sees all fragmentation, allocation and replication, the DDBMS is said to have no distribution transparency. The user need to refer to specific fragment copies by appending the site name to the relation.

In systems with high degree of distribution transparency, the user sees a single integrated schema with no details of fragmentation, replication or distribution. The DDBMS stores all the details in the distribution catalog.

---

Query processing in DDBMS

Query processing involves communication costs. Consider the following DDBMS

SITE 1
- EMPLOYEE (LNAME, SSN, ...)
  - 10,000 records, each 100 bytes
  - LNAME 20 bytes
  - SSN 9 bytes

SITE 2
- DEPARTMENT (DNAME, DNUMBER, MGRSSN, MGRSTARTDATE)
  - 100 records, each 35 bytes long
  - DNAME 10 bytes
  - MGRSSN 9 bytes

SITE 3
Query: for each department retrieve the DNAME and the LNAME of the department manager.

---

Query processing

strategy 1
Transfer both EMPLOYEE and DEPARTMENT relations to site 3 and perform join at site 3.
Communication cost =

Strategy 2
Transfer EMPLOYEE relation to site 2, perform join and send the result to site 3.
Size of result =
Transfer cost =

Strategy 3
Transfer DEPARTMENT relation to site 1, perform join and send the result to site 3.
Transfer cost =
Query processing using semijoin

Suppose the result is required at site 2, we have two strategies
1. transfer EMPLOYEE to site 1 and perform join
   - How many employee tuples would participate in the join?
   - Which are those?
   - The idea of semijoin is to find and transfer only those tuples.

2. transfer DEPARTMENT to site 1, perform join and transfer the result back to site 2

The semijoin between two relations $R$ and $S$ is defined as

$$R \times_{A=B} S = \pi_{<R>}(R \bowtie_{A=B} S)$$

In a distributed environment this is implemented by taking the projection $F = \pi_B(S)$ and transferring it to the site where $R$ resides and performing the join.

The transfer cost for our example is =

Is $R \times S$ same as $S \times R$?

Concurrent control and recovery in DDBMS

Concurrent control and recovery problems in a DDBMS include the following,

- dealing with multiple copies of the data
to maintain consistency, handle recovery of a site after failure
- handle failure of individual sites
- handle failure of communication links
- deal with distributed deadlock
There are two broad techniques used for distributed concurrency control.

- one copy of the data item is designated as the distinguished copy and the locks for this data item are associated with the distinguished copy
- locking is accomplished in a distributed manner after voting

**Concurrency control based on distinguished copy of a data item**

There are a number of variations of this scheme. 1. primary site technique: This is an extension of the centralized approach.

All locks are kept at one site and all lock requests are sent to that site.

The site becomes a coordinator site or primary site

Once a Trans. obtains a read_lock on an item, it can access any copy of the item.

If a Trans. obtains a write_lock on an item and updates the item, then what happens to other copies of the item?

**Disadvantages**

Since all lock requests have to be processed at one site, the site gets overloaded and may cause system bottleneck.

Failure of the primary site paralyzes the system, hence sacrificing the system reliability and availability

A variation of this scheme is to have a backup of the primary site

2. Primary copy technique

In this technique the distinguished copies of different items are stored at different sites.

The task of lock operations is distributed hence no bottle neck.

If one site fails, only some transactions are affected. Which ones?

In all the cases, if the coordinator site fails, a new coordinator need be chosen. This procedure will have to take care of aborting the transactions etc.
Distributed concurrency control based on voting

This method is truly distributed with the decision being made by all sites

- To obtain a lock on a data item, the lock request is sent to each of the sites having a copy of the data item
- Each copy can grant or deny the request
- If a majority of sites grant the lock, then the transaction has the lock on the item. It will then transmit this information to all the sites
- If a majority is not reached within a certain timeout period, the request is cancelled and all sites so informed.

Distributed recovery

Recovery in distributed databases is more complicated

Consider the simple case when site X sends a message to site Y and expects a reply. If a reply is not received within a certain amount of time, there are four possibilities.

1. The message never made it to Y
2. Message reached Y, but Y is dead
3. Y received the message, but did not respond
4. Y responded, but response was lost

Unless additional information is provided, or messages sent it is difficult to determine what exactly happened.