Advanced DBMS

Unit-2

Parallel Databases

- Introduction
- I/O Parallelism
- Interquery Parallelism
- Intraquery Parallelism
- Intraoperation Parallelism
- Interoperation Parallelism
- Introduction to Distributed Databases

Introduction

- Parallel machines are becoming quite common and affordable
  - Prices of microprocessors, memory and disks have dropped sharply
  - Recent desktop computers feature multiple processors and this trend is projected to accelerate
- Databases are growing increasingly large
  - large volumes of transaction data are collected and stored for later analysis.
  - multimedia objects like images are increasingly stored in databases
- Large-scale parallel database systems increasingly used for:
  - storing large volumes of data
  - processing time-consuming decision-support queries
  - providing high throughput for transaction processing

Parallelism in Databases

- Data can be partitioned across multiple disks for parallel I/O.
- Individual relational operations (e.g., sort, join, aggregation) can be executed in parallel
    - data can be partitioned and each processor can work independently on its own partition.
- Queries are expressed in high level language (SQL, translated to relational algebra)
    - makes parallelization easier.
- Different queries can be run in parallel with each other.   Concurrency control takes care of conflicts.
- Thus, databases naturally lend themselves to parallelism.

I/O Parallelism

- Reduce the time required to retrieve relations from disk by partitioning
- The relations on multiple disks.
- Horizontal partitioning – tuples of a relation are divided among many disks such that each tuple resides on one disk.
- Partitioning techniques (number of disks = *n*):

  Round-robin:

  Send the *I* th tuple inserted in the relation to disk *i* mod *n*.

  Hash partitioning:

  - Choose one or more attributes as the partitioning attributes.
  - Choose hash function *h* with range 0…*n* - 1
  - Let *i* denote result of hash function *h* applied to the partitioning attribute value of a tuple. Send tuple to disk *i*.

I/O Parallelism (Cont.)

- Partitioning techniques (cont.):
- **Range partitioning**:
  - Choose an attribute as the partitioning attribute.
  - A partitioning vector [v_o, v_1, ..., v_{n-2}] is chosen.
  - Let v be the partitioning attribute value of a tuple. Tuples such that $v_i \leq v_{i+1}$ go to disk I + 1. Tuples with $v < v_0$ go to disk 0 and tuples with $v \geq v_{n-2}$ go to disk n-1.

    E.g., with a partitioning vector [5,11], a tuple with partitioning attribute value of 2 will go to disk 0, a tuple with value 8 will go to disk 1, while a tuple with value 20 will go to disk2.

Interquery Parallelism

- Queries/transactions execute in parallel with one another.
- Increases transaction throughput; used primarily to scale up a transaction processing system to support a larger number of transactions per second.
- Easiest form of parallelism to support, particularly in a shared-memory parallel database, because even sequential database systems support concurrent processing.
- More complicated to implement on shared-disk or shared-nothing architectures
    - Locking and logging must be coordinated by passing messages between processors.
    - Data in a local buffer may have been updated at another processor.
    - **Cache-coherency** has to be maintained — reads and writes of data in buffer must find latest version of data.

Intraquery Parallelism

- Execution of a single query in parallel on multiple processors/disks; important for speeding up long-running queries.
- Two complementary forms of intraquery parallelism:
    - **Intraoperation Parallelism** – parallelize the execution of each individual operation in the query.
    - **Interoperation Parallelism** – execute the different operations in a query expression in parallel.

the first form scales better with increasing parallelism because
the number of tuples processed by each operation is typically more than the number of operations in a query.

Introduction to Distributed Databases

- A distributed database is basically a database that is not limited to one system, it is spread over different sites, i.e, on multiple computers or over a network of computers.
- A distributed database system is located on various sites that don’t share physical components. This may be required when a particular database needs to be accessed by various users globally. It needs to be managed such that for the users it looks like one single database.

Distributed Databases-Types

Homogeneous Database:

In a homogeneous database, all different sites store database identically. The operating system, database management system, and the data structures used – all are the same at all sites. Hence, they’re easy to manage.

Distributed Databases-Types

Heterogeneous Database:

In a heterogeneous distributed database, different sites can use different schema and software that can lead to problems in query processing and transactions. Also, a particular site might be completely unaware of the other sites. Different computers may use a different operating system, different database application. They may even use different data models for the database. Hence, translations are required for different sites to communicate.

Distributed Databases-Data Storage

1. Replication –

In this approach, the entire relationship is stored redundantly at 2 or more sites. If the entire database is available at all sites, it is a fully redundant database. Hence, in replication, systems maintain copies of data.

Distributed Databases-Data Storage

This is advantageous as it increases the availability of data at different sites. Also, now query requests can be processed in parallel.

However, it has certain disadvantages as well. Data needs to be constantly updated. Any change made at one site needs to be recorded at every site that relation is stored or else it may lead to inconsistency. This is a lot of overhead. Also, concurrency control becomes way more complex as concurrent access now needs to be checked over a number of sites.

Distributed Databases-Data Storage

2. Fragmentation –

In this approach, the relations are fragmented (i.e., they’re divided into smaller parts) and each of the fragments is stored in different sites where they’re required. It must be made sure that the fragments are such that they can be used to reconstruct the original relation (i.e, there isn’t any loss of data).

Fragmentation is advantageous as it doesn’t create copies of data, consistency is not a problem.

Distributed Databases-Data Storage

Fragmentation of relations can be done in two ways:

Horizontal fragmentation – Splitting by rows –

The relation is fragmented into groups of tuples so that each tuple is assigned to at least one fragment.

Vertical fragmentation – Splitting by columns –

The schema of the relation is divided into smaller schemas. Each fragment must contain a common candidate key so as to ensure a lossless join.