Database System Architectures
Parallel Database

- Distributed Data base system consists of loosely coupled sides that share no physical Components and the parallel processors are tightly coupled and constitute a single data Base system

Scope

- distributed database are widely used in large data processing, today’s word using the internet, e–banking system, whether forecasting etc. where large amount of data is processed, so there we need of data processing if data is distributed in many places then that is distributed data processing, there fore scope of parallel data bases and distributed data processing is very bright.

Research

- Lots of research are going on in distributed data processing and parallel databases.

Parallel Databases

- Introduction
- I/O Parallelism
- Interquery Parallelism
- Intraquery Parallelism
- Intraoperation Parallelism
- Interoperation Parallelism
- Design of Parallel Systems

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 $[v_o, v_1, \ldots, v_{n-2}]$ 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.

Comparison of Partitioning Techniques

- Evaluate how well partitioning techniques support the following types of data access:

  1.Scanning the entire relation.

  2.Locating a tuple associatively – **point queries**.

  - E.g., $r.A = 25$.

  3.Locating all tuples such that the value of a given attribute lies within a specified range – **range queries**.

  - E.g., $10 \leq r.A < 25$.

Comparison of Partitioning Techniques (Cont.)

Round robin:

- Advantages
  - Best suited for sequential scan of entire relation on each query.
  - All disks have almost an equal number of tuples; retrieval work is thus well balanced between disks.
- Range queries are difficult to process
  - No clustering -- tuples are scattered across all disks

Comparison of Partitioning Techniques(Cont.)

Hash partitioning:

- Good for sequential access
  - Assuming hash function is good, and partitioning attributes form a key, tuples will be equally distributed between disks
  - Retrieval work is then well balanced between disks.
- Good for point queries on partitioning attribute
  - Can lookup single disk, leaving others available for answering other queries.
  - Index on partitioning attribute can be local to disk, making lookup and update more efficient
- No clustering, so difficult to answer range queries

Comparison of Partitioning Techniques (Cont.)

- Range partitioning:
- Provides data clustering by partitioning attribute value.
- Good for sequential access
- Good for point queries on partitioning attribute: only one disk needs to be accessed.
- For range queries on partitioning attribute, one to a few disks may need to be accessed
    - Remaining disks are available for other queries.
    - Good if result tuples are from one to a few blocks.
    - If many blocks are to be fetched, they are still fetched from one to a few disks, and potential parallelism in disk access is wasted
        - Example of execution skew.

Partitioning a Relation across Disks

- If a relation contains only a few tuples which will fit into a single disk block, then assign the relation to a single disk.
- Large relations are preferably partitioned across all the available disks.
- If a relation consists of *m* disk blocks and there are *n* disks available in the system, then the relation should be allocated **min**(m,n) disks.

Handling of Skew

- The distribution of tuples to disks may be **skewed** — that is, some disks have many tuples, while others may have fewer tuples.
- **Types of skew:**
  - **Attribute-value skew.**
    - Some values appear in the partitioning attributes of many tuples; all the tuples with the same value for the partitioning attribute end up in the same partition.
    - Can occur with range-partitioning and hash-partitioning.
  - **Partition skew.**
    - With range-partitioning, badly chosen partition vector may assign too many tuples to some partitions and too few to others.
    - Less likely with hash-partitioning if a good hash-function is chosen.

Handling Skew in Range-Partitioning

- To create a balanced partitioning vector (assuming partitioning attribute forms a key of the relation):
  - Sort the relation on the partitioning attribute.
  - Construct the partition vector by scanning the relation in sorted order as follows.
    - After every $1/n^{th}$ of the relation has been read, the value of the partitioning attribute of the next tuple is added to the partition vector.
  - *n* denotes the number of partitions to be constructed.
  - Duplicate entries or imbalances can result if duplicates are present in partitioning attributes.
- Alternative technique based on **histograms** used in practice

Handling Skew Using Virtual Processor Partitioning

- Skew in range partitioning can be handled elegantly using **virtual processor partitioning**:
  - create a large number of partitions (say 10 to 20 times the number of processors)
  - Assign virtual processors to partitions either in round-robin fashion or based on estimated cost of processing each virtual partition
- Basic idea:
  - If any normal partition would have been skewed, it is very likely the skew is spread over a number of virtual partitions
  - Skewed virtual partitions get spread across a number of processors, so work gets distributed evenly!

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.

Cache Coherency Protocol

- Example of a cache coherency protocol for shared disk systems:
  - Before reading/writing to a page, the page must be locked in shared/exclusive mode.
  - On locking a page, the page must be read from disk
  - Before unlocking a page, the page must be written to disk if it was modified.
- More complex protocols with fewer disk reads/writes exist.
- Cache coherency protocols for shared-nothing systems are similar. Each database page is assigned a *home* processor. Requests to fetch the page or write it to disk are sent to the home processor.

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