Fundamentals of Database Systems

[Parallel and Distributed Databases]

Malay Bhattacharyya

Assistant Professor

Machine Intelligence Unit
Indian Statistical Institute, Kolkata

November, 2019

Basics

Why parallelism?

Basics

Why parallelism?

- Speedup: Queries can be executed faster because more resources, such as processors and disks, are becoming available (at low cost!!!).
- Scaleup: Increasing workloads can be handled without increased response time, via an increase in the degree of parallelism.

Basics

Why parallelism?

- Speedup: Queries can be executed faster because more resources, such as processors and disks, are becoming available (at low cost!!!).
- Scaleup: Increasing workloads can be handled without increased response time, via an increase in the degree of parallelism.

The parallelism of a database depends on the architecture of the system.

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 can be expressed in high level languages (SQL, translated to relational algebra) to make the parallelization easier.
- Different queries can be run in parallel with each other where the concurrency control takes care of the conflicts that might occur.

Parallelism of I/O – Different partitioning techniques

- Round-robin: Send the i^{th} tuple of the relation to disk $i\%n$.

Parallelism of I/O – Different partitioning techniques

- Round-robin: Send the i^{th} tuple of the relation to disk $i\%n$.
- Hash partitioning: Choose one or more attributes as the partitioning attributes. Choose hash function h with range $0 \ldots n-1$. If i denotes the result of hash function h applied to the partitioning attribute value of a tuple, then send the corresponding tuple to disk i.

Parallelism of I/O – Different partitioning techniques

- Round-robin: Send the i^{th} tuple of the relation to disk $i\%n$.
- Hash partitioning: Choose one or more attributes as the partitioning attributes. Choose hash function h with range $0 \ldots n-1$. If i denotes the result of hash function h applied to the partitioning attribute value of a tuple, then send the corresponding tuple to disk i.
- Range partitioning: Choose an attribute as the partitioning attribute. A partitioning vector $[v_0, v_1, \ldots, v_{n-2}]$ 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 $n-1$.

Comparing partitioning techniques – Metrics

How well partitioning techniques support the following types of data access:

1. Scanning the entire relation.
2. Point query – locating a tuple associatively, e.g., $r.A = 25$.
3. Range query – locating all tuples such that the value of a given attribute lies within a specified range, e.g., $10 \leq r.A < 25$.

Comparing partitioning techniques

Round robin:

1. 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.
2. Disadvantages: Range queries are difficult to process. No clustering – tuples are scattered across all disks.

Comparing partitioning techniques

Hash partitioning:

1. Advantages: 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
2. Disadvantages: No clustering, so difficult to answer range queries

Comparing partitioning techniques

Range partitioning:

1. Advantages: 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.

Skew problem

The distribution of tuples to disks may be skewed – this means some disks have many tuples, while others may have fewer tuples.

Skew problem

The distribution of tuples to disks may be skewed – this means some disks have many tuples, while others may have fewer tuples.

Types of skew:

1. 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. It can occur with range-partitioning and hash-partitioning.
2. 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.

Skew problem

The distribution of tuples to disks may be skewed – this means some disks have many tuples, while others may have fewer tuples.

Types of skew:

1. 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. It can occur with range-partitioning and hash-partitioning.
2. 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.

Note: The problem of skewness can be solved using histograms.

Interquery and intraquery parallelism

- Interquery parallelism – Queries/transactions execute in parallel with one another.
- Intraquery parallelism – Execution of a single query in parallel on multiple processors/disks.

Interquery and intraquery parallelism

- Interquery parallelism – Queries/transactions execute in parallel with one another.
- Intraquery parallelism – Execution of a single query in parallel on multiple processors/disks.

Intraquery parallelism can have two complementary forms – intraoperation (parallelize the execution of each individual operation in the query) and interoperation (execute the different operations in a query expression in parallel) parallelism.

Advantages and disadvantages of interquery parallelism

Advantages:

1. Increases transaction throughput.
2. Used primarily to scale up a transaction processing system to support a larger number of transactions per second.
3. Easiest form of parallelism to support, particularly in a shared-memory parallel database, because even sequential database systems support concurrent processing.

Advantages and disadvantages of interquery parallelism

Advantages:

1. Increases transaction throughput.
2. Used primarily to scale up a transaction processing system to support a larger number of transactions per second.
3. Easiest form of parallelism to support, particularly in a shared-memory parallel database, because even sequential database systems support concurrent processing.

Disadvantages:

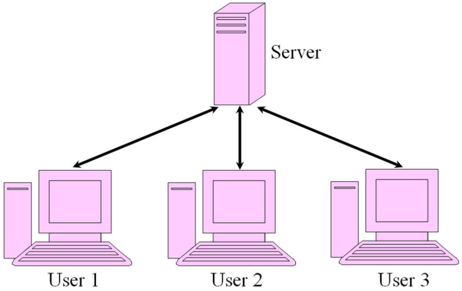
1. More complicated to implement on shared-disk or shared-nothing architectures.

Advantages and disadvantages of intraquery parallelism

Advantages:

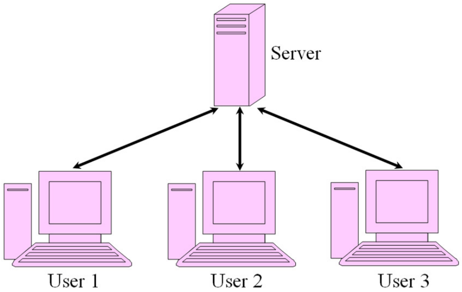
1. This is important for speeding up long-running queries.
2. Intraoperation parallelism scales better with increasing parallelism.

Basics



Centralized client-server architecture

Basics



Centralized client-server architecture

A distributed database system consists of loosely coupled sites that share no physical component. Database systems that run on each site are independent of each other. Transactions may access data at one or more sites.

Homogeneous and heterogeneous databases

In a homogeneous distributed database
– all sites have identical software
– all are aware of each other and agree to cooperate in processing user requests
– each site surrenders part of its autonomy in terms of right to change schemas or software
– the entire system appears as a single system to the user

Homogeneous and heterogeneous databases

In a homogeneous distributed database
– all sites have identical software
– all are aware of each other and agree to cooperate in processing user requests
– each site surrenders part of its autonomy in terms of right to change schemas or software
– the entire system appears as a single system to the user

In a heterogeneous distributed database
– different sites may use different schemas and software
– difference in schema is a major problem for query processing
– difference in software is a major problem for transaction processing

Data distribution

Data can be distributed in two ways:

- Replication – The system maintains several identical replicas (copies) of the relation, and stores each replica at a different site. The alternative to replication is to store only one copy of a relation.
- Fragmentation – The system partitions the relation into several fragments, and stores each fragment at a different site.

Data distribution

Data can be distributed in two ways:

- Replication – The system maintains several identical replicas (copies) of the relation, and stores each replica at a different site. The alternative to replication is to store only one copy of a relation.
- Fragmentation – The system partitions the relation into several fragments, and stores each fragment at a different site.

Note: The fragmentation can be lossless (original relation can be restored from the partitions) or lossy (original relation can not be restored from the partitions).

Data distribution

Replication	**Fragmentation**
Advantageous in terms of high availability	Might not be readily available
Advantageous in terms of time complexity but not space complexity	Maintains a balance between the time and space complexity
Disadvantageous in view of the redundancy and for updating	No redundancy or problem in updating

Data transparency

Data transparency denotes the degree to which a system user may remain unaware of the details of how and where the data items are stored in a distributed system.

Data transparency

Data transparency denotes the degree to which a system user may remain unaware of the details of how and where the data items are stored in a distributed system.

It can be of the following types:

1. Replication transparency – Users are not required to know what data objects have been replicated, or where replicas have been placed.
2. Fragmentation transparency – Users do not have to be concerned with how a relation has been fragmented.
3. Location transparency – Users are not required to know the physical location of the data.

Horizontal fragmentation

Name	Age	Area
Malay	36	Crowdsourcing
Debapriyo	39	Information Retrieval

Name	Age	Area
Debasis	59	Reliability
Amitava	60	Inference

Horizontal fragmentation

Name	Age	Area
Malay	36	Crowdsourcing
Debapriyo	39	Information Retrieval

Name	Age	Area
Debasis	59	Reliability
Amitava	60	Inference

Note: Horizontal fragmentation is lossless when union of the fragments produces the original relation.

Vertical fragmentation

Name	Age
Malay	36
Debapriyo	39
Debasis	59
Amitava	60

Area
Crowdsourcing
Information Retrieval
Reliability
Inference

Vertical fragmentation

Name	Age
Malay	36
Debapriyo	39
Debasis	59
Amitava	60

Area
Crowdsourcing
Information Retrieval
Reliability
Inference

Note: Vertical fragmentation is lossless when natural join of the fragments produces the original relation.

Advantages of horizontal and vertical fragmentation

Horizontal:

- It allows parallel processing on fragments of a relation.
- It allows a relation to be split so that tuples are located where they are most frequently accessed.

Advantages of horizontal and vertical fragmentation

Horizontal:

- It allows parallel processing on fragments of a relation.
- It allows a relation to be split so that tuples are located where they are most frequently accessed.

Vertical:

- It allows tuples to be split so that each part of the tuple is stored where it is most frequently accessed.
- Here tuple-id attribute allows efficient joining of vertical fragments.
- It allows parallel processing on a relation.

Advantages of horizontal and vertical fragmentation

Horizontal:

- It allows parallel processing on fragments of a relation.
- It allows a relation to be split so that tuples are located where they are most frequently accessed.

Vertical:

- It allows tuples to be split so that each part of the tuple is stored where it is most frequently accessed.
- Here tuple-id attribute allows efficient joining of vertical fragments.
- It allows parallel processing on a relation.

Vertical and horizontal fragmentation can be mixed (hybrid fragmentation) and the advantage is that fragments may be successively fragmented to an arbitrary depth.

Hybrid fragmentation

A hybrid fragment neither include all the tuples for an attribute (likewise vertical fragmentation) nor all the attributes for a tuple (likewise horizontal fragmentation).

Name	Age
Malay	36
Debapriyo	39
Debasis	59

Distributed transaction management

- Transaction may access data at several sites.

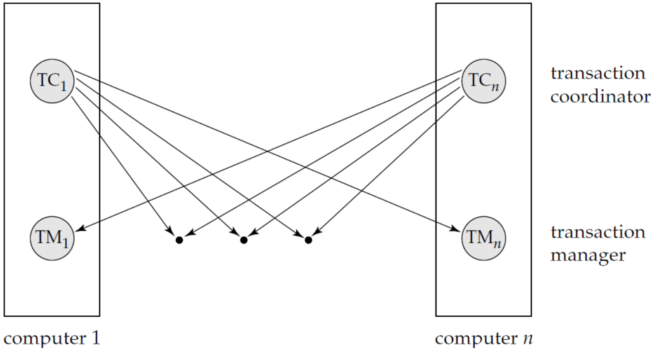
Distributed transaction management

- Transaction may access data at several sites.
- Each site has a local *transaction manager* responsible for:
  1. Maintaining a log for recovery purposes
  2. Participating in coordinating the concurrent execution of the transactions executing at that site.

Distributed transaction management

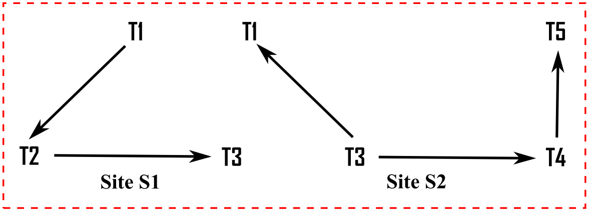
- Transaction may access data at several sites.
- Each site has a local *transaction manager* responsible for:
  1. Maintaining a log for recovery purposes
  2. Participating in coordinating the concurrent execution of the transactions executing at that site.
- Each site has a *transaction coordinator*, which is responsible for:
  1. Starting the execution of transactions that originate at the site.
  2. Distributing subtransactions at appropriate sites for execution.
  3. Coordinating the termination of each transaction that originates at the site, which may result in the transaction being committed at all sites or aborted at all sites.

Distributed transaction management

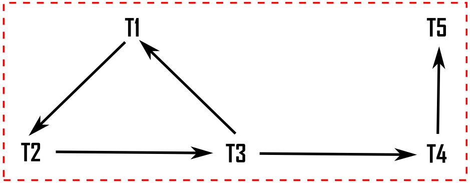


Distributed system architecture for transaction management

Distributed deadlock handling



Local view



Global view

Local and global wait-for graphs