More Database Models: distributed databases

November 7, 2023

In a centralised database management system all system components and data reside at a single site
In recent years, there has been a growing trend towards distributed database systems.

- Distributed nature of some database applications: a company may have many different locations and data at each location
- Increased reliability and availability: if data and DBMS s/w is distributed, then user is not dependent on just one site
- Data Sharing
- Improved Performance: local queries may be executed more efficiently.

- Distribution leads to increased complexity in system design and implementation:
- Additional functionality is required:
  - to allow access to remote sites
  - to keep track of data distribution data and replication
  - to devise execution strategies for queries and transactions that access many sites
  - to maintain consistency across sites
  - to allow recovery from new types of failures

Data Fragmentation

Horizontal Fragmentation

Fragment tables into sub-tables based on certain SELECT restrictions.

Vertical Fragmentation

Sets of attributes from a table are stored at different sites. This type of fragmentation can be defined by a PROJECT operation.
Vertical fragmentation is never totally disjoint as the key attributes must be stored at each site. Necessary in order to reconstruct the table.

Hybrid fragmentation

Defined by a sequence of SELECTs and PROJECTs

Fragmentation Schema

A fragmentation schema (which will be in every data catalog for each client) is a full set of fragmentation definitions.

Allocation Schema

An allocation schema (also in the catalog) defines the location of fragments

Replication

- Useful in improving availability of data
- Full Replication: store whole database at each site
- Partial Replication: replicate certain fragments
- No replication

Query processing

- In a centralised DBMS, we attempt to maximise efficiency by reducing the size of intermediate tables.
- In distributed DBMS, the most significant measure of cost is the quantity of data transferred.
- The most common execution strategies are based on reducing network traffic.
- The $semi - join$ “operator”, is the standard approach where no redundant tuples or attributes are transferred. Only attributes needed in join conditions or in the final result are transferred.

Distributed Query Processing: small example

- Assume relations *employee*, *dept*, and *project* are stored at $site_1$, $site_2$ and $site_3$ respectively.
- Assume also no fragmentation or replication of these relations.
- We wish to join tables to obtain the result at some site $site_i$, i.e we need to compute employee $\bowtie$ dept $\bowtie$ project.

- No one strategy is always the best.
- The relations involved, their size, selectivity of joins etc. will all vary over time.

Distributed Query Processing: semi-join

- The semi-join operator is a commonly adopted approach to guarantee some degree of efficiency.
  Let relations r and s be at $site_1$ and $site_2$ respectively. We wish to calculate $r \bowtie s$.
  Often, there will be many tuples in r and s that will not be included in the result.

Distributed Query Processing: semi-join

1. Create $tmp1$ comprising the join attributes of r
2. Ship $tmp1$ to $site_2$
3. Execute $s \bowtie tmp1 : -tmp2$ at $site_2$
4. Ship $tmp2$ to $site_1$
5. Evaluate $r \bowtie tmp2$ at $site_1$

Usually reduces the number of spurious tuples to be transferred.

Concurrency Control and Recovery

Concurrency Control and Recovery

Numerous problems arise in distributed databases that do not exist in a centralised DBMS:

- Dealing with multiple copies of data items. The concurrency control mechanism must ensure consistency between these items
- Failure of individual sites: The DBMS should continue to operate; and when the site recovers it should be brought up to date
- Failure of communication links
- Distributed Commit
- Distributed Deadlock

Recovery

- In order to facilitate recovery we must generate atomicity of transactions.
- This becomes a more difficult problem in distributed databases as a transaction must commit at all sites or must fail at all sites.
- A two-phase commit procedure is usually adopted.
- A transaction coordinator is located at one site, $site_i$
- When a transaction T completes execution coordinator is informed.

Phase 1

- [*prepare*, *T*] is added to log at $site_i$; log is force-written.
- [*prepare*, *T*] message is sent by the coordinator to all involved sites.
- Transaction managers at sites return an [*abort*, *T*] message or a [*ready*, *T*] message (whether T has successfully terminated or not).
- If [*ready*, *T*] entry is sent by a site, individual logs are then force-written.

Phase 2

- if all sites respond with a [*ready*, *T*] (within in given time), the transaction is considered committed.
- [*commit*, *T*] is added to the log. Force-write log.
- else, [*abort*, *T*] is placed. Force-write log.
- Coordinator then informs all sites as to whether T has committed or not.
- Variations on this approach. Most of these variations attempt to increase the efficiency of recovery.

- These algorithms require a coordinator. The coordinator is usually chosen in advance. Measures have to be taken to ensure correctness if the coordinator happens to crash.
- Backup Coordinator: maintains up-to-date copy of coordinator. Can be extended to have a chain of backups.
- Election protocols: If the coordinator crashes, any involved sites may try to assume control. If they obtain the majority of votes, they assume control and inform all others.

Concurrency Control

- Most approaches merely extend centralised approaches of 2-phase locking and time-stamping:
- With locking a single Lock Manager can be chosen: one site chosen as lock manager. All locks are granted by the lock manager at this site.
- Advantage: Easy to implement.
- Disadvantages: Leads to bottleneck at lock manager site; particularly if fine-grained locking used.
- Over-dependence on one site

Multiple lock managers

- Each site possesses its own lock manager for items present at that site.
- For non-replicated items, no real problems arise.
- For items replicated at many sites, a transaction issuing a *write_item* needs to send a request to all lock managers. Each lock manager sends an acceptance or a rejection.
- A majority protocol is typically used, i.e., if the majority of responses are grants, then transaction obtains lock.

Distributed Deadlock

- One potential problem of distributed locking protocols is the possibility of distributed deadlock.
- Further complicated by the potential of phantom deadlocks.
- Many algorithms exist to try and efficiently deal with this problem.

Timestamping

- Time-stamping can also be extended:
- Timestamps generated at each local site
- Difficulties arise with respect to ordering transactions. If some sites have higher throughput, they will have higher timestamps and hence timestamp ordering will be invalid.
- Usually create timestamp by actualy taking combining actual timestamp and site identifier.
- Ordering is usually enforced by using logical clock schemes.