CHAPTER 19

Distributed Databases

Distributed databases in general, and *heterogeneous* distributed databases in particular, are of increasing practical importance, as organizations attempt to integrate databases across physical and organizational boundaries. Such interconnection of databases to create a distributed or multidatabase is in fact proving crucial to competitiveness for many companies. This chapter reconsiders the issues addressed earlier in the text, such as query processing, recovery and concurrency control, from the standpoint of distributed databases.

This is a long chapter, and is appropriate only for an advanced course. Single topics may be chosen for inclusion in an introductory course. Good choices include distributed data storage, heterogeneity and two-phase commit.

Exercises

19.16 Discuss the relative advantages of centralized and distributed databases.

Answer:

- A distributed database allows a user convenient and transparent access to data which is not stored at the site, while allowing each site control over its own local data. A distributed database can be made more reliable than a centralized system because if one site fails, the database can continue functioning, but if the centralized system fails, the database can no longer continue with its normal operation. Also, a distributed database allows parallel execution of queries and possibly splitting one query into many parts to increase throughput.
- A centralized system is easier to design and implement. A centralized system is cheaper to operate because messages do not have to be sent.

19.17 Explain how the following differ: fragmentation transparency, replication transparency, and location transparency.

Answer:

a. With fragmentation transparency, the user of the system is unaware of any fragmentation the system has implemented. A user may formulate queries against global relations and the system will perform the necessary transformation to generate correct output.

b. With replication transparency, the user is unaware of any replicated data. The system must prevent inconsistent operations on the data. This requires more complex concurrency control algorithms.

c. Location transparency means the user is unaware of where data are stored. The system must route data requests to the appropriate sites.

19.18 When is it useful to have replication or fragmentation of data? Explain your answer.
Answer: Replication is useful when there are many read-only transactions at different sites wanting access to the same data. They can all execute quickly in parallel, accessing local data. But updates become difficult with replication. Fragmentation is useful if transactions on different sites tend to access different parts of the database.

19.19 Explain the notions of transparency and autonomy. Why are these notions desirable from a human-factors standpoint?
Answer: Autonomy is the amount of control a single site has over the local database. It is important because users at that site want quick and correct access to local data items. This is especially true when one considers that local data will be most frequently accessed in a database. Transparency hides the distributed nature of the database. This is important because users should not be required to know about location, replication, fragmentation or other implementation aspects of the database.

19.20 If we apply a distributed version of the multiple-granularity protocol of Chapter 15 to a distributed database, the site responsible for the root of the DAG may become a bottleneck. Suppose we modify that protocol as follows:

- Only intention-mode locks are allowed on the root.
- All transactions are given all possible intention-mode locks on the root automatically.

Show that these modifications alleviate this problem without allowing any nonserializable schedules.
Answer: Serializability is assured since we have not changed the rules for the multiple granularity protocol. Since transactions are automatically granted all intention locks on the root node, and are not given other kinds of locks on it, there is no need to send any lock requests to the root. Thus the bottleneck is relieved.

19.21 Study and summarize the facilities that the database system you are using provides for dealing with inconsistent states that can be reached with lazy propagation of updates.
Answer:

PostgreSQL does not have built-in support for replication. However, there are several external projects that add replication support to the database engine.

- **Slony-I** adds basic master-slave replication functionality to PostgreSQL with updates to the replicated databases performed lazily. Slony-I uses the trigger mechanism of PostgreSQL to implement replication. A consistent view of the replicated data is provided by preserving transactions across the replicated sites. Thus, the replicated sites will always have a consistent, although potentially older, version of the data.
  Slony-I does not support multi-master replication, and therefore does not have to deal with inconsistent states due to updates at multiple sites.
- **Postgres-R** extends PostgreSQL to add synchronous replication, thus avoiding the consistency issues of performing lazy updates. However, Postgres-R is experimental software, and not ready for production use.

Several databases support lazy multi-master replication, which involves the risk of inconsistent data. The first problem lies in detecting the conflict, and the second problem lies in resolving the conflict. Key values and timestamps are the primary means of detecting conflicts:

- Uniqueness conflicts are detected when there are duplicates on a primary key.
- Update conflicts are detected when two sites update the same item independently. Timestamps or version numbers sent with updates are used to detect such conflicts, by recording the timestamp/version number prior to and after the update.
- Delete conflicts occur when a transaction at one site deletes a tuple, which another site concurrently updates (before the delete propagated to that site).

Oracle provides several options for resolving conflicts:

- Latest time stamp: Most recent update wins
- Overwrite: Overwrites current value with new value, without checking for conflict.
- Discard: Ignores the value
- Additive: Difference of the two values is added to the current value (current value = current value + (new value - old value)). Alternatives include retaining the minimum or maximum value.

As an alternative to using the predefined conflict resolution policies in Oracle, administrators can create stored procedures to resolve each type of conflict on a relation.

In Microsoft SQL Server 2008, multi-master replication is called peer-to-peer replication. SQL Server supports conflict detection between all combinations of insert, delete and update, based on the primary key, and a node identifier plus a version number stored with each tuple. In case conflicts are detected, the default is to stop replication, but the default can be overridden using stored procedures to resolve conflicts.

In addition, SQL Server also supports a form of replication called Merge Replication, which is based on a single-publisher/multiple-subscriber model, where each subscriber can update data and send updates back to the publisher. The resolution model simply retains one of the conflicting updates, with the publisher having higher priority, and a priority scheme between subscribers used to choose the winning subscriber in case two subscribers updated the same data.

19.22 Discuss the advantages and disadvantages of the two methods that we presented in Section 19.5.2 for generating globally unique timestamps.

Answer: The centralized approach has the problem of a possible bottleneck at the central site and the problem of electing a new central site if it goes down. The distributed approach has the problem that many messages must be exchanges to keep the system fair, or one site can get ahead of all other sites and dominate the database.

19.23 Consider the relations:

$$\begin{array}{l} \textit{employee}\ (\textit{name, address, salary, plant\_number}) \\ \textit{machine}\ (\textit{machine\_number, type, plant\_number}) \end{array}$$

Assume that the *employee* relation is fragmented horizontally by *plant_number*, and that each fragment is stored locally at its corresponding plant site. Assume that the *machine* relation is stored in its entirety at the Armonk site. Describe a good strategy for processing each of the following queries.

a. Find all employees at the plant that contains machine number 1130.

b. Find all employees at plants that contain machines whose type is "milling machine."

c. Find all machines at the Almaden plant.

d. Find employee $\bowtie$ machine.

Answer:

a. i. Perform $\Pi_{plant\_number}\ (\sigma_{machine\_number=1130}\ (machine))$ at Armonk.

ii. Send the query Π_{name} $(employee)$ to all site(s) which are in the result of the previous query.

iii. Those sites compute the answers.

iv. Union the answers at the destination site.

b. This strategy is the same as in part a of this exercise, except the first step should be to perform

$$\Pi_{plant\_number}\ (\sigma_{type=\text{"milling machine"}}\ (machine))\ \text{at Armonk.}$$

c. i. Perform $\sigma_{plant\_number\ =\ x}\ (machine)$ at Armonk, where x is the plant _number for Almaden.

ii. Send the answers to the destination site.

d. Strategy 1:

i. Group *machine* at Armonk by plant number.

ii. Send the groups to the sites with the corresponding plant _number.

iii. Perform a local join between the local data and the received data.

iv. Union the results at the destination site.

Strategy 2:

Send the *machine* relation at Armonk, and all the fragments of the *employee* relation to the destination site. Then perform the join at the destination site.

There is parallelism in the join computation according to the first strategy but not in the second. Nevertheless, in a WAN the amount of data to be shipped is the main cost factor. We expect that each plant will have more than one machine, hence the result of the local join at each site will be a cross-product of the employee tuples and machines at that plant. This cross-product's size is greater than the size of the *employee* fragment at that site. As a result the second strategy will result in less data shipping, and will be more efficient.

19.24 For each of the strategies of Exercise 19.23, state how your choice of a strategy depends on:

a. The site at which the query was entered.

b. The site at which the result is desired.

Answer:

a. Assuming that the cost of shipping the query itself is minimal, the site at which the query was submitted does not affect our strategy for query evaluation.

b. For the first query, we find out the plant numbers where the machine number 1130 is present, at Armonk. Then the employee tuples at all those plants are shipped to the destination site. We can see that this strategy is more or less independent of the destination site. The same can be said of the second query. For the third query, the selection is performed at Armonk and results shipped to the destination site. This strategy is obviously independent of the destination site.
For the fourth query, we have two strategies. The first one performs local joins at all the plant sites and their results are unioned at the destination site. In the second strategy, the *machine* relation at Armonk as well as all the fragments of the *employee* relation are first shipped to the destination, where the join operation is performed. There is no obvious way to optimize these two strategies based on the destination site. In the answer to Exercise 19.23 we saw the reason why the second strategy is expected to result in less data shipping than the first. That reason is independent of destination site, and hence we can in general prefer strategy two to strategy one, regardless of the destination site.

19.25 Is the expression $r_i \ltimes r_j$ necessarily equal to $r_j \ltimes r_i$? Under what conditions does $r_i \ltimes r_j = r_j \ltimes r_i$ hold?
Answer: In general, $r_i \ltimes r_j \neq r_j \ltimes r_i$. This can be easily seen from Exercise 19.11, in which $r \ltimes s \neq s \ltimes r$. $r \ltimes s$ was given in 19.11, while

$s \ltimes r =$

C	D	E
3	4	5
3	6	8
2	3	2

By definition, $r_i \ltimes r_j = \Pi_{R_i}(r_i \bowtie r_j)$ and $r_j \ltimes r_i = \Pi_{R_j}(r_i \bowtie r_j)$, where R_i and R_j are the schemas of r_i and r_j respectively. For $\Pi_{R_i}(r_i \bowtie r_j)$ to be always equal to $\Pi_{R_j}(r_i \bowtie r_j)$, the schemas R_i and R_j must be the same.

19.26 If a cloud data-storage service is used to store two relations r and s and we need to join r and s, why might it be useful to maintain the join as a materialized view? In your answer, be sure to distinguish among various meanings of "useful": overall throughput, efficient use of space, and response time to user queries.
Answer: Performing a join on a cloud data-storage system can be very expensive, if either of the relations to be joined is partitioned on attributes

other than the join attributes, since a very large amount of data would need to be transferred to perform the join. However, if $r \bowtie s$ is maintained as a materialized view, it can be updated at a relatively low cost each time each time either r or s is updated, instead of incurring a very large cost when the query is executed. Thus, queries are benefitted at some cost to updates.

Considering the various notions of usefulness, with the materialized view, overall throughput will be much better if the join query is executed reasonably often relative to updates, but may be worse if the join is rarely used, but updates are frequent.

The materialized view will certainly require extra space, but given that disk capacities are very high relative to IO (seek) operations and transfer rates, the extra space is likely to not be an major overhead since a large number of disks are needed anyway to handle the IO load and data transfer load.

The materialized view will obviously be very useful to evaluate join queries, reducing time greatly by avoiding a large amount of data transfer across machines.

19.27 Why do cloud-computing services support traditional database systems best by using a virtual machine instead of running directly on the service provider's actual machine?

Answer: By using a virtual machine, if a physical machine fails, virtual machines running on that physical machine can be restarted quickly on one or more other physical machines, improving availability. (Assuming of course that data remains accessible, either by storing multiple copies of data, or by storing data in an highly available external storage system.)

19.28 Describe how LDAP can be used to provide multiple hierarchical views of data, without replicating the base-level data.

Answer: This can be done using referrals. For example an organization may maintain its information about departments either by geography (i.e. all departments in a site of the the organization) or by structure (i.e. information about a department from all sites). These two hierarchies can be maintained by defining two different schemas with department information at a site as the base information. The entries in the two hierarchies will refer to the base information entry using referrals.