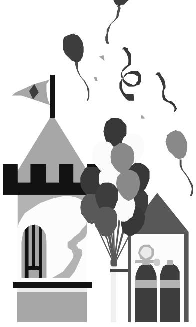
XIV. Multidatabases

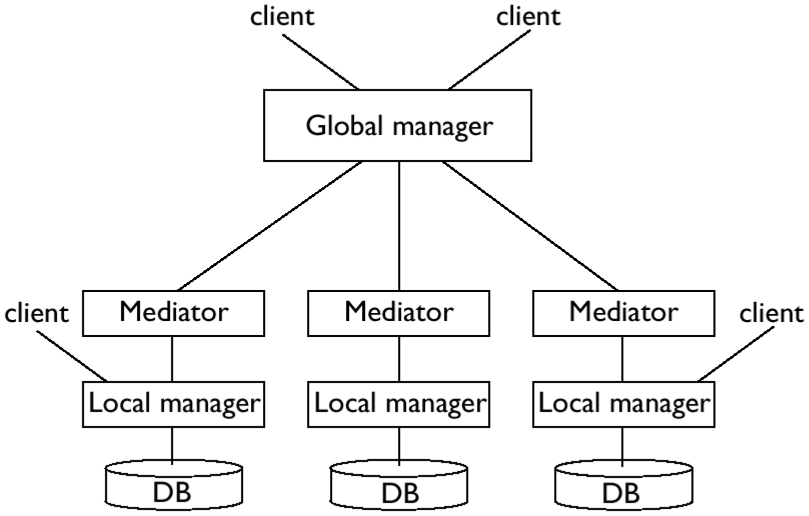
Multidatabases
Parallel Database Systems
Replidated Databases
Research Directions



Multidatabases

- Each participating database continues to be used by its respective users (programs or end users.)
- Systems are also accessed by modules, called ***mediators***, which show only the portion of a database that is to be exported; mediators rely on a ***global manage***r, to carry out the integration.
- In general, data cannot be modified by the mediators, because each source system is autonomous.
- Features:
  - ✓ presents an integrated view to the users, as if the databases were integrated;
  - ✓ provides a high level of transparency;
  - ✓ currency is also high, because data is accessed at source;

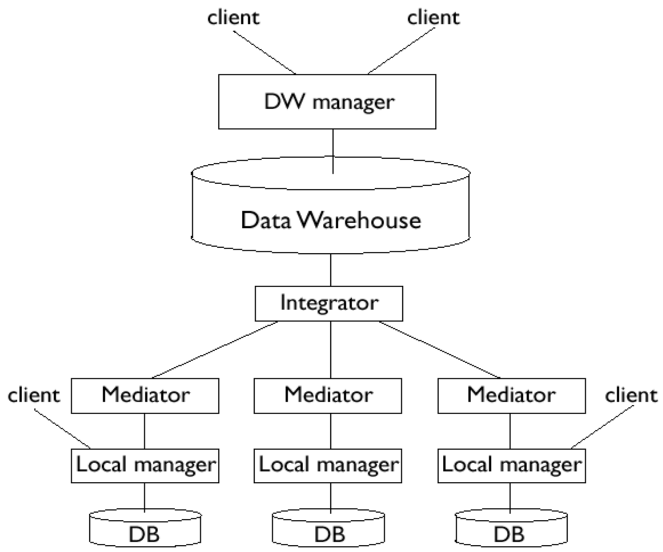
A Multidatabase System



Replicated Data Architectures

- They guarantee read-only access to secondary copies of a database.
- An example of replication is a ***data warehouse***, which downloads periodically data extracted from various heterogeneous distributed systems for purposes of analysis and decision support.
- Key feature: offers a high level of integration and transparency, but a reduced degree of currency.

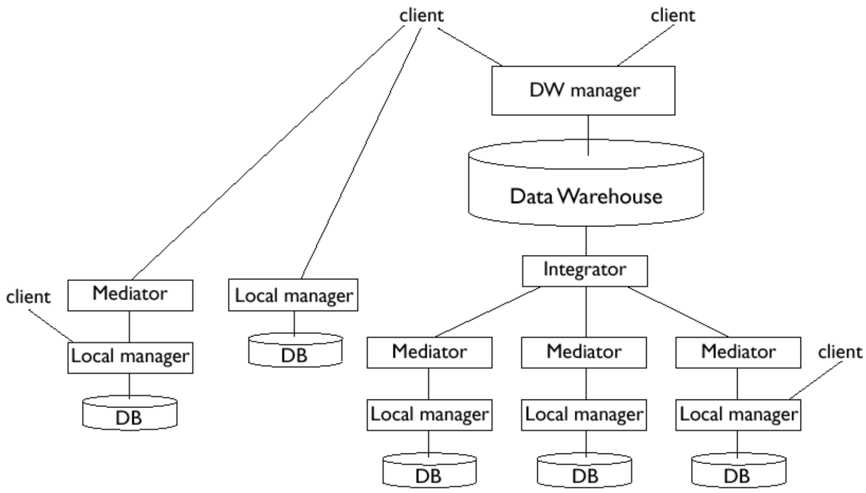
Systems Based on Replicated Data



Application-Level Data Integration

- Data integration is carried out explicitly by the application
- In the next example, three sources are integrated: an external database, a local database and a data warehouse, which in turn uses three sources of information
- Features:
  - ✓ Low degree of transparency and integration, with a degree of currency that depends on specific demands

Example of Application-Level Integration



Parallel Databases

- Were developed in the nineties along with the spread of standard multiprocessor architectures, after the failure of specialized database architectures (the so-called ***database machines***) during the eighties.
- Parallelism is possible with multiprocessor architectures both with and without shared memory.
- Parallelism in databases is useful because many data management operations are repetitive in nature, and can be carried out in parallel with great efficiency.
- For example, a complete scan of a large database can be done using *n* scans, each on a portion of the database. If the database is stored on *n* different disks managed by *n* different processors, resulting in a speedup of approximately 1/*n* of the time required for a serial scan,

Inter-Query Parallelism

- Parallelism is called ***inter-query*** when it carries out several queries in parallel.
  - ✓ The load imposed on the DBMS is typically characterized by simple and frequent transactions (up to thousands of transactions per second.)
  - ✓ Parallelism is introduced by multiplying the number of servers and allocating an optimal number of requests to each server.
  - ✓ In many cases, the queries are redirected to servers by a ***dispatcher*** process.
  - ✓ Useful for OLTP systems.

Intra-Query Parallelism

- Parallelism is known as ***intra-query*** when it executes parts of the same query in parallel:
  - ✓ The load on the DBMS is characterized by a few extremely complex queries, which are decomposed into various partial sub-queries, to be executed in parallel.
  - ✓ In general, queries are carried out one after another, using the entire multi-processor system for each query.
  - ✓ Useful for OLAP systems.

Parallelism and Data Fragmentation

- Parallelism for databases requires some form of data fragmentation: the fragments are distributed among many processors and allocated to distinct secondary memory devices.
- Consider:

```
ACCOUNT(AccNum, Name, Balance)
TRANSACTION(AccNum,Date,SerialNumber,
  TransType,Amount)
```

Fragmentation based on predefined intervals of account number

Example of a Typical OLTP Query

- A typical OLTP query with inter-query parallelism:

```
procedure Query5(:acc-num, :total);
select Balance into :total
  from Account
  where AccNum = :acc-num;
end procedure;
```

- Directed towards specific fragments depending on their selection predicates

Example of a Typical OLAP Query

- A typical OLAP query with intra-query parallelism:

```
procedure Query6();
select AccNum, sum(Amount)
  from Account join Transaction
 on Account.AccNum = Transaction.AccNum
  where Date >= 1.1.1998
     and Date < 1.1.1999 group by AccNum
  having sum(Amount) > 100000;
end procedure;
```

- Carried out on all of the fragments in parallel

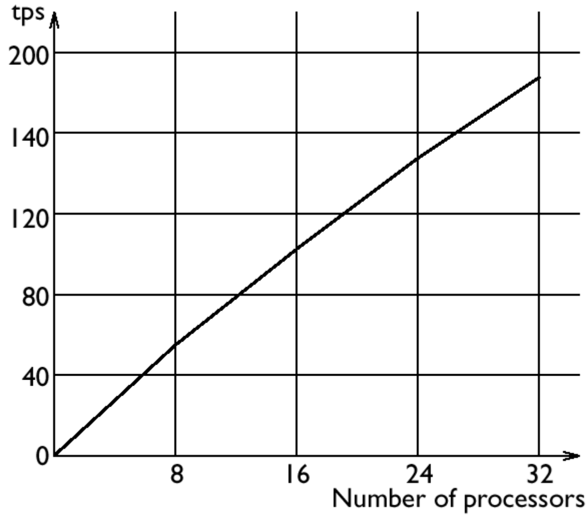
Distributed Joins

- The join of pairs of fragments corresponding to the same account number interval; the joins between the matching fragments can be carried out in parallel.
- Essential for intra-query parallelism: The parallel execution of *n* joins on fragments of dimension (1/*n*) is obviously preferable to the execution of a single join that involves the entire table
- In general, when the initial fragmentation does not allow the distributed execution of the joins present in the query, data is dynamically redistributed to support distributed joins.

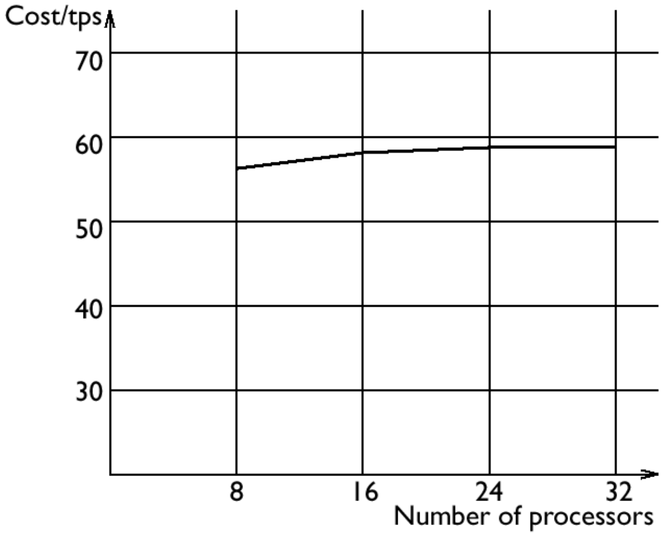
Speed-Up and Scale-Up

- The ***speed-up*** curve characterizes only inter-query parallelism and measures the increase of services, measured in ***tps*** (transactions per second), against the increase in the number of processors; In the ideal situation, services increase almost linearly against the increase in processors.
- The ***scale-up*** curve characterizes both inter-query parallelism and intra-query parallelism, and measures the average cost of a single transaction against the increase of the number of processors; In the ideal situation, the average costs remain almost constant with an increase in processors. In such cases, we say that the system ‘scales well’.

A Typical Speed-Up Curve



A Typical Scale-Up Curve



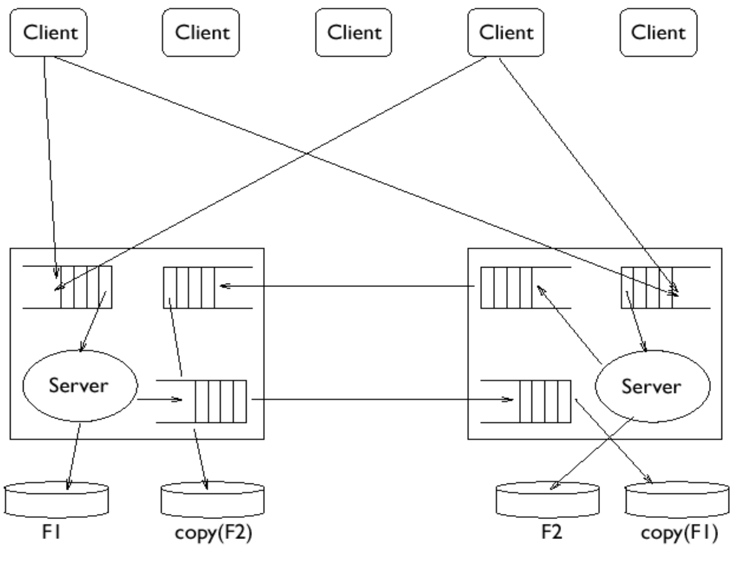
Transaction Benchmarks

- Specific tests for measuring the efficiency of (possibly parallel) DBMS; used to produce speed-up and scale-up curves under standard workload conditions.
- Standardized by TPC (Transaction Processing Performance Council), a committee of about thirty suppliers of DBMSs and transaction systems
- Three main benchmarks (TPC-A, TPC-B and TPC-C) respectively for OLTP, mixed, and OLAP applications. Can refer to a mainframe-based, client-server, or parallel architecture
- Parameters of benchmarks: transaction code, size of the database, the method used for generating data, the distribution of the arrivals of transactions, also the techniques for measuring and auditing the benchmark

Replicated Databases

- Data replication is an essential service for the creation of many distributed applications
- Provided by products called ***data replicators***, whose function is to maintain consistency among copies. They operate transparently to applications running on the DBMS server
- In general, there is one ***main copy*** and various ***secondary copies***, and updates are propagated asynchronously (without the two-phase commit protocol)
- Propagation is *incrementa*l when it is based on data variations, sent from the main copy to the secondary copy
- The use of replication makes a system less sensitive to failure, because if the main copy is unavailable it is possible to use one of its copies

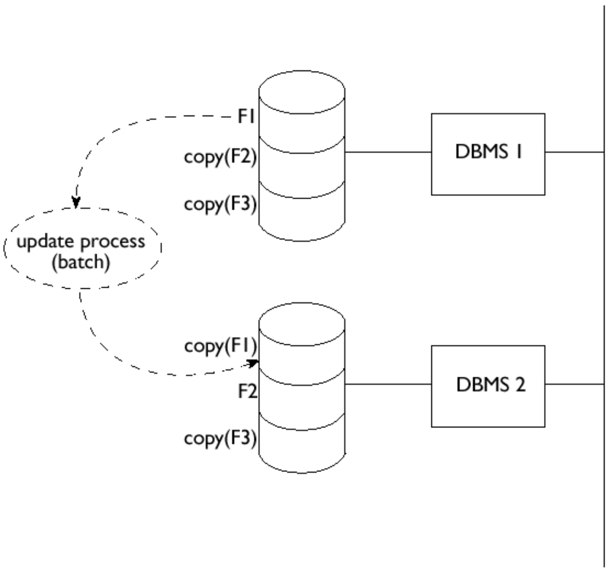
Typical Architecture for Data Replication



Discussion of the Architecture

- The architecture has two identical sites, each managing the entire database; half is the main copy and the other half is the secondary copy.
- All transactions are sent to the main copy and then possibly redirected to the secondary copy.
- Each 'access point' to the system is connected to both sites.
- In the case of a failure for one site, the system commutes almost instantly all the transactions to the other site, which is powerful enough to sustain the entire load.
- When the problem is resolved, the replication manager restores the data transparently and then resets the two sites to normal operations.
- Specialized for high availability.

Example Architecture for Data Replication



The Tandem Information System

- An application created by Tandem in the mid-eighties; had ten factories in various parts of the world, each responsible for the production of a specific part of the architecture of a computer.
- Tables representing the available parts in the company were fragmented to reflect the physical distribution of the parts, and then allocated to the nodes, co-located with a factory, in a redundant way: the main copy of each fragment was on the node responsible for the production process of the parts described in that fragment, while secondary copies were replicated in all other nodes
- The replication manager acted periodically, by collecting a batch of modifications on a given fragment and applying them asynchronously to all the other fragments

Advanced Replication Mechanisms

- ***Symmetrical replication*** allows modifications on any copy, with a 'peer-to-peer' situation among the copies:
  - ✓ It is possible to introduce conflicts, in that two copies of the same information are managed in a concurrent way *without* concurrency control;
  - ✓ Techniques are capable of revealing inconsistencies after their occurrence; repair is application-specific.
- ***Disconnected replication***: used with *mobile* systems, in which the connection with the database can be broken; for example, a salesperson can connect to the database in order to download data on merchandise and upload orders received. The salesperson is normally disconnected from the database and accepts transactions on the copy; the copy is 'reconciled' with the main copy at the end of the sale activity.

Research Directions for Multidatabases

- Consider a company database:

  `Cust(name,addr,phone)`

  `Sales(custName,prod#,price,amount,date)`

  `Prod(prod#,name,price,inStock)`

- Each salesperson leaving for a trip downloads parts of the `Prod`, `Sales` and `Cust` relations. On their trip, they update customer, and sales information.
- Each of these databases evolves autonomously from the original, and there is no global manager. However, we'd like to enforce **coordination rules**, such as:

  "Updates to a customer address must be propagated to other databases"

Coordination Rules

- These are inter-database soft constraints. They are checked every time there is an updated to one of the relevant databases, and are enforced through some protocol.
- Examples:

  ✓ `Master:Cust(n,x) and SalesN:Cust(n,y)`

  `—> x=y)                propagate last`

  /* the latest addr is propagated to the other database */

  ✓ `Master:Prod(p#,p) and SalesN:Prod(p#,p')`

  `—> p=p') propagate (Master—>SalesN)`

  /* the Master copy prices are always propagated to the other databases, not the other way around */

More Coordination Rules

✓ `Maria:TravelB=x and Paolo:TravelB=y and Fausto:TravelB=z —> x+y+z≤15MLit`

`equi-distribute`

/* if their total budget is x > 15MLit, reduce each budget by (x - 15ML)/3 */

✓ `Master:Prod(p#,n) and SalesN:Prod(p#,n')`

`undo`

/* no updates allowed to product names */

✓ `Master:Prod(p#,.) == SalesN:Prod(p#,.)`

`propagate (Master->SalesN)`

/* propagate added or deleted Product tuples in the Master database */

Compatibility Rules

- They define correspondences between different databases at two levels:
  - ✓ Constant to constant, e.g., `'one'` — `'uno'`;
  - ✓ Relational signature to relational signature, e.g.,

    `Cust(name,addr) — Customer(nm,ad)`
- Of course, such correspondences can be one-to-one, one-to-many or many-to-many.

A Data Model for Multidatabases

- A Multidatabase system consists of one or more databases and a set of coordination rules.
- Operations that can be performed on the system include:
  - ✓ Add or delete a database;
  - ✓ Update or query a database;
  - ✓ Add, delete or update a coordination rule.
- Each database ‘knows’ (‘is acquainted with’) only with databases it shares coordination and compatibility rules.
- Queries are local; global queries can be expressed by using compatibility rules:

  “Give me all customers in all acquainted databases”

  fetches customers from all databases in the transitive closure of the aquaintaince relation wrt this database.

Research Problems

- A formal semantics to the Multidatabase Model (note: hard problem, standard Logic techniques assume a global model which serves as interpretation of whatever you are trying to formalize.)
- Efficient global query processing techniques (possibly exploiting parallelism.)
- A formal transaction model for coordination rules, supporting ‘soft’ enforcement mechanisms.
- Efficient implementation techniques for coordination rule enforcement.
- …more...