REGULAR ENTRY IN
ENCYCLOPEDIA OF DATABASE SYSTEMS
http://refworks.springer.com/database-systems/

DISTRIBUTED ARCHITECTURE

Tore Risch, Uppsala University, Sweden, http://user.it.uu.se/~torer/

SYNONYMS

Parallel database, federated database, multi-database, peer-to-peer database

DEFINITION

A distributed database is a database where data management is distributed over several nodes (computers) in a computer network. In a central DBMS the data is managed by one node whereas in a distributed DBMS the data is managed by several nodes. A *distributed DBMS* is a database manager consisting of several nodes distributed over a network. Each node is a database manager by itself that communicates with other nodes in the network. In a regular distributed DBMS it is up to the database administrator to manually specify how data collections (e.g. relational tables) are distributed over the nodes when a distributed database is designed. Queries and updates to the distributed relations are transparently translated by the distributed DBMS into data operations on the affected nodes giving the user the impression of using a single database, called *query* and *update transparency*. Thus the distributed DBMS provides distribution transparency for database users but not for the database administrator.

Closely related to distributed DBMSes are *parallel databases* where a parallel DBMS engine runs on usually a cluster. The parallel DBMS automatically determines how data structures are internally distributed over the nodes providing distribution transparency also for the database administrator, called *schema transparency*.

The purpose of *heterogeneous databases* is to be able to combine data from several independently developed autonomous databases. Heterogeneous databases can be divided into federated databases, mediators, and multi-databases. In a *federated database* the database administrator defines a single *global integration schema* describing how data in underlying databases are mapped to the integration schema view. This provides distribution transparency for integrated data. *Mediators* allow the definition of several views over data from different data sources. Since it may be difficult to define integration schemas and views when there are many participating autonomous databases, *multi-databases* relax the distribution transparency also for the database users who there specify queries and updates using a multi-database query language where individual data collections in the participating nodes can be explicitly referenced.

A related technology is *peer-to-peer systems* where networks of files are distributed over the Internet. Meta-data is associated with the files and the user can search for files satisfying conditions. Peer-to-peer search is usually made by propagating queries between the peers. The consistency and correctness of queries are relaxed compared to regular databases in order to provide better performance and node autonomy.

HISTORICAL BACKGROUND

Distributed DBMSs were pioneered by System R and Ingres* in the beginning of the 80-ies. Early distributed DBMSs assumed slow communication between nodes having limited amounts of main memory geographically distributed in a wide area network. The database administrator instructed the distributed DBMS where to place data, while the user could specify transparent queries to the distributed DBMS without detailed knowledge of where data was placed.

The evolvement of computer clusters provided hardware resources for very high performing database servers running on clusters, *parallel databases*. Since the communication between cluster nodes is very fast and not geographically distributed, the database administrator need not provide manual placement rules of distributed data, i.e. the parallel DBMS provides full distribution transparency also for the database administrator. With the evolvement of fast wide area computer networks parallel DBMS technology can be used also for some geographically distributed databases. However, it should be noted that update latency has to be taken into account for large geographical distances because of the speed of light. In general geographically distributed databases still requires manual distribution.

Not least the development of the Internet has caused the need to integrate data from many pre-existing databases. The area of *heterogeneous databases* deals with tools and methodologies to combine data from several autonomous databases. While distributed and parallel databases assumed all data managed by one distributed DBMS, heterogeneous databases integrate databases using different DBMS and different schemas.

There are several flavors of heterogeneous databases:

- *Federated databases* require the definition of a *global integration schema* containing mappings to the participating databases' schemas. The federated database becomes a central server on top of the participating autonomous databases.
- As the number of databases to integrate increases it becomes very difficult or impossible to define a global integration schema over the large numbers of autonomous databases. *Multi-databases* provide no global conceptual schemas and instead a *multi-database query language* allows specification of queries searching through many participating databases.
- Mediators provide a middle ground between a single integration schema and no schema at all. Instead the user can define mediator views that combine and reconcile data from different data sources. Such views require a query language that can express queries over several databases, i.e. a multi-database query language. The mediator system becomes a middleware between users and wrapped data sources.

While distributed databases could handle transparent queries and updates for a small number of nodes, the evolvement of the Internet requires technologies to deal with geographically distributed databases having 1000s of nodes. *Peer-to-peer systems* enable such highly distributed file access where users search for data stored in peers. In a *peer-to-peer database* queries are propagated between the participating peer nodes. To improve performance at the expense of query correctness the propagation may stop after a certain number of hops. This is sufficient for many modern applications that do not have strict consistency requirements; for example Internet search engines do not guarantee the full correctness of answers.

SCIENTIFIC FUNDAMENTALS

The architectures of DDBMSs can be classified along different dimensions. The following table classifies different kinds of distributed DBMS architectures:

	Autonomy	Schema transparency	Query transparency	Update transparency	Naming transparency	Central schema
Parallel	no	Yes	Yes	Yes	Yes	Yes
Regular Distributed	no	No	Yes	Yes	Yes	Yes
Federated	yes	No	Yes	Limited	Yes	Yes
Mediators	yes	No	Yes	Limited	No	No
Multi-databases	yes	No	No	No	No	No
Peer-to-peer	Yes	No	Yes	Yes	Yes	No

Autonomy and Heterogeneity

Different distributed DBMS architectures provide different levels of autonomy for the participating nodes.

A *homogeneous* distributed database is a distributed database where all nodes are managed by the same distributed DBMS. A homogeneous distributed database can be regarded as a central database distributed over many nodes where data and processing is internally transparently distributed over several nodes. By contrast, a heterogeneous database is a (distributed or central) database where data originates from participating autonomous databases possibly using different DBMSs.

Regular distributed and *parallel* databases are homogeneous. One distributed DBMS manages all data. *Distributed database design* involves designing the schema in a top-down fashion as for a conventional central database. Parallel databases provide automatic and transparent data placement without user intervention, while regular distributed databases require the database administrator to specify how data should be distributed over nodes. In regular distributed and parallel database the nodes have no autonomy at all.

Federated databases are central database servers that integrate data from several participating databases. Federated databases are thus heterogeneous. *Global integration schemas* are defined that integrate data originated in the participating databases. The design of the integrated schema needs to deal with data integration issues on how to combine the same or similar data represented differently in different participating databases. Different participating databases may use different DBMSs. The schemas of the participating databases are designed before the integrated database schema is designed. Thus the design process for heterogeneous databases becomes bottom-up, whereas homogeneous databases are usually designed top-down. The design of the integrated schema needs to deal with data integration issues on how to combine the same or similar data represented differently in different participating databases.

Both *federated databases*, *mediators*, and *multi-databases* are heterogeneous. The main difference between them is how integration schemas are defined. Federated database assume one global integration schema. If there are many different participating databases it is difficult to define such a global integration schema. This is relaxed in mediators, which allow the definition of many integration schemas as views over wrapped underlying data sources of different kinds.

In multi-databases the user is given access to a *multi-database query language* where he can specify queries over many sources. A multi-database query language provides the basis for defining mediator views.

Finally, the aim of peer-to-peer databases is distributed queries in a widely distributed network of heterogeneous nodes. Unlike parallel and distributed databases the individual nodes are not managed by a single system, but independently.

Transparency

Distributed databases can be classified according to what kinds of transparency they provide w.r.t. distribution of data. Three different kinds of transparency can be identified for different kinds of services provided by the distributed DBMS, *schema transparency*, *query transparency*, and *update transparency*.

Schema transparency means that the distributed DBMS decides completely on its own where to place data on different nodes. The database administrator has the impression of using a single database and specifies the logical schema without considering any distribution at all. However, often it is desirable to allow the database administrator to specify how to distribute data, and thus relax schema transparency. For example, for performance and to allow local control, a geographically distributed database for a large enterprise may need to cluster employee data according to the countries where departments are located. Therefore full schema transparency is often provided only on local area networks or cluster computers where the communication between nodes is very fast.

With *query transparency* the distribution of data is not reflected in user queries. Queries are transparently translated by a distributed query optimizer into queries and updates to the affected nodes giving the user the impression of using a single database. By analyzing a given user query the distributed query optimizer can often statically determine which nodes to access. Query execution plans can execute in parallel on different nodes with partial results transported between nodes and combined on other nodes. Query transparency is very important for distributed databases since it is very difficult and error prone to manually implement distributed communicating execution plans.

Update transparency allows database updates to be specified without taking distribution into account. A distributed transaction manager propagates updates to affected nodes. Distributed or parallel DBMS provide update transparency.

In the classification above, only *parallel DBMSs* provide complete transparency for everyone using the database, database administrators as well as users. The term *regular distributed database* refers to a distributed DBMS with query and update transparency but without schema transparency.

With *naming transparency* users are provided with a single name for a distributed relation defined in terms of several internal relations stored on separate nodes. Regular distributed, parallel, federated, and peer-to-peer databases provide naming transparency, which is relaxed for mediators and multi-databases.

Distributed database design involves manual specification to the distributed DBMS of the distribution of data collections. The database administrator can tune the data placement in a wide area computer network. The two fundamental methods for such manual data distribution are *fragmentation* and *replication.* Fragmentation splits a collection (e.g. table) into separate non-overlapping segments on different nodes, while replication stores identical copies of a collection on different nodes. The distributed DBMS guarantees that queries and updates of fragmented or replicated collections are transparent so the user need not be aware of how data is distributed.

Fragmentation (or partitioning) allows the administrator of a distributed database to manually specify on which nodes the DBMS should place different sections of each distributed data collection. In a distributed relational database tables are fragmented. For example, the placement of employee records in a relation can be fragmented according to in which countries different employees work. Fragmentation speeds up database queries and updates since it allows parallel access to distributed fragments. Furthermore, by analyzing queries and updates the query optimizer can often determine exactly which nodes are affected and send the query/update statements only to those nodes.

Replication allows the DBA to declare to the DDBMS to place the same data collections on more than one node. For example, a relational table may be replicated on several nodes. Replication speeds up data access at the expense of update cost. However, as explained below, if consistency is relaxed the update cot may be reduced.

Federated databases also provide query and update transparency by allowing the database administrator to define a *global integration schema* that hides the underlying integrated databases.

Mediators provide some query transparency by allowing users to define views over integrated databases. Update transparency is more problematic as it requires updatable views.

With *multi-databases* transparency is further relaxed so the user can reference individual databases explicitly in queries and updates.

Finally, *peer databases* provide query and update transparency in widely distributed systems but do not require fully correct query answers.

Consistency

If data is widely distributed over many nodes in a network the cost of maintaining data consistency may be very high. The transaction manager must guarantee that all transactions are atomic and updates propagated to affected nodes so that the database is kept consistent. Two and three phase commit protocols are needed when more than one node is affected by an update to guarantee full update transparency. These protocols are expensive when many nodes are involved and relaxed update transparency may suffice to enable higher transaction performance. If the same kind data is present on many nodes updates must be propagated to all replicas, which can be very expensive in a geographically distributed database.

Regular distributed databases usually provide transaction atomicity as an option. However, because of the high cost of transaction atomicity modern distributed DBMS also provide the option to propagate updates lazily, thus compromising the consistency.

In a parallel DBMS running on a cluster, the nodes inside the cluster run DBMS kernel software which is completely controlled by the parallel DBMS. From the user’s point of view it looks like a central DBMS; the main difference being the higher performance provided by parallelization of DBMS kernel software.

In regular distributed and parallel DBMSs a single database kernel manages all distributed data. All individual nodes are running the same distributed DBMS software. Different nodes may have different roles, e.g. some nodes handle query processing, some nodes handle locking, some nodes handle recovery, etc. The DBMS is a monolithic systems distributed over several nodes controlling the consistency of the individual nodes.

In general consistent updates are difficult to achieve with heterogeneous databases since the participating databases are autonomous and the integrating DBMS may not have access to transaction managers of the participating databases.

In peer-to-peer databases the data consistency is relaxed for higher update and query performance. Data can be partly replicated for efficiency but the system does not guarantee consistency among the replicas so updates need not always be propagated to all replicas at every update. This means that queries may return less reliable result, which is often acceptable in a widely distributed database. This is similar to how search engines compromise query quality for performance.

Distributed catalog management

A particular problem for distributed databases is how and where to handle catalog data, such as the overall schema, statistics about data collections, the location of data collections, and how data collections are replicated and partitioned. The catalog information is accessed intensively by database users in queries and updates. On the other hand, in most DBMSs it is assumed that schema and catalog information changes slowly, which, for example, permits pre-compilation of (distributed) database queries. The assumption that catalog data changes slowly but is intensively accessed is a case for replicating catalog information on many nodes, in particular on those coordinating nodes with which the users interact. On the other hand, in a heterogeneous database with many participating autonomous nodes, the assumption that schemas and data placements do not change usually does not hold.

Regular distributed and parallel databases assume few participating non-autonomous nodes and the catalog is therefore replicated. Federated databases have a central architecture where all interaction with the database is through the global schema and it contains replications of catalog information from the participating databases. For mediators, multi-databases, and peer-to-peer there is no central global schema and the query processing nodes are autonomous. Therefore the catalogue data cannot be fully replicated and it will be up to different nodes to cache catalog data when needed. The validity of cached catalog data needs to be properly handled though; otherwise queries may fail or even return the wrong data.

CROSS REFERENCES
Distributed databases
Parallel databases
Peer-to-peer databases
Data integration
Mediators

RECOMMENDED READING

[1] M. T. Özsu and P. Valduriez: *Principles of Distributed Database Systems*, 2nd ed., Prentice Hall, 1999
[2] A. P. Sheth and J. A. Larson. Federated database systems for managing distributed, heterogeneous, and autonomous databases. ACM Computer Surveys, 22(3):183-235, September 1990.
[3] W. Litwin, L. Mark and N. Roussopoulos, Interoperability of Multiple Autonomous Databases ACM Computing Surveys, Vol. 22, No. 3, September 1990
[4] Wiederhold, G., Mediators in the architecture of future information systems. IEEE Computer, Vol 25, No 3, 1992.
[5] Special Section on Peer-to-peer-based Data Management, IEEE Transactions on Knowledge and Data Engineering, Vol. 16, No. 7, July 2004