Transaction Model for Federated Databases Based on DOK[*]

TEREFE Muleta LI Ruixuan

School of Computer Science and Technology, Huazhong University of Science & Technology, Wuhan 430074, P.R.China.

(email: terefemu@hotmail.com rxli2001@hotmail.com)

Abstract This paper defines architecture and frame work of Distributed Object Kernel (DOK) that federated systems may be built. Distributing control and decision making allows the user to exploit most of the available parallelism within this type of system in order to provide scalability. Concurrency control is provided by time stamping and transaction classes in autonomous, distributed and heterogeneous databases.

Keywords: Transaction management, Federated databases, Deadlock, Object kernel, Wrapper.

1. Introduction

A federated database [1] is a virtual database layer on top of a collection of autonomous source databases. It provides a uniform integrated view on the total collection of source databases. In general, two major design approaches are available for federating databases:

- global schema federated databases,
- federated databases language systems.

The former approach takes the independently developed local schema's, resolves semantic and syntactic differences and creates an integrated summary of all information of the union of the local schema's. This approach is also known as view integration. The latter approach uses one language to access all local databases. The federated language must incorporate therefore all basic features of every local database language.

This paper defines architecture and frame work that federated systems may be built. Concurrency control is provided by time stamping and transaction classes. Although serializability is supported (as a degenerate case), this architecture is designed to support non-traditional applications that can exploit semantic-based concurrency control and non-serial transaction histories.

[*] Supported by National High Performance Fund. Terefe Muleta , male, master, research directions: federated database systems.

2. Federated database system's architecture based on DOK

The issues in Distributed Object Kernel (DOK) system are discussed from two perspectives: physical and logical. The DOK system is responsible for global management of cooperative database systems (CDBS). Its physical architecture is illustrated in Figure 1 and consists of three layers: *query layer, transaction layer* and *wrapper layer*. Each of the first two layers consists of sub-layers responsible for more specific tasks. The wrapper layer provides the upper Layers with an interface to interact with local databases. We assume that network connectivity between heterogeneous platforms (local database systems) is provided by middleware software (like CORBA [2]).

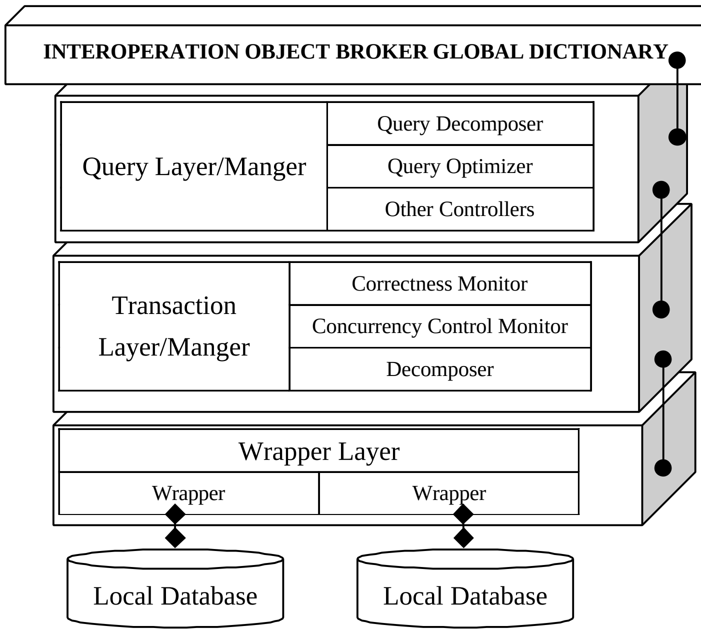

Figure.1. The DOK Physical Architecture

Database Agents: The DOK lower layer is the *database/agent layer* that consists of a set of agents that provide typical database services (such as, the list of users who can access a given information). These agents include the *component database agent*, the *user agent*, the *data agent,* and the *constraint agent*.

The next generation of multi-databases is called *cooperative database systems* (CDBS). They allow customized autonomy using appropriate software agents which are capable of enforcing a required level of security, and appropriate mechanisms to understand the underlying semantics of local database applications in order to improve the components of cooperative processing (e.g. query optimization). CDBSs provide a flexible mechanism for different levels of autonomy requirements. For example, a company can make some of its information public, some private and the rest public but in control. A database system should be able to dynamically and transparently change its behavior in terms of global information access, task negotiations, and global transaction processing, according to different autonomy requirements of the enterprise.

Task Agents: The DOK middle layer is the *task layer* which involves agents that specialize in specific functions that are required for cooperative processing. For the security enforcement task, the agents of this layer ensure that all the security aspects of processing are carried out properly so that global security can always be preserved. These agents include a *security processor*, *query processor*, *database version manager*, *global transaction manager*, and *constraint manager*. The *security processor* has a major role in assisting a DOK coordinator in enforcing global security. The *query processor* takes a query that violates any global constraints and modifies it so that it no longer violates the constraints. The *database version manager* and *version agent* keep track of previously modified data. Finally, the *query processor* collates the results of the global query and presents it to the user. However, before it does so, it needs to verify the results that it received. This involves making joins or unions of the individual results that were collected.

Intelligent Wrappers: To manage different types of databases, the DOK system relies on a set of wrappers. The DOK wrappers provide translation functions enabling the mapping of conventional databases into Reflective Distributed Object Model (RDOM). The wrappers also deal with advanced functions of negotiations and communications enabling the DOK system to comprehend the semantics hidden in local database applications, to identify potential systems to perform specific tasks, to negotiate the execution of the tasks. The DOK wrappers are in charge of enforcing different levels of autonomy, thus allowing customized approach to cooperation. A wrapper may involve different agents, and one of them will specialize in enforcing security both at the global and local levels. By adapting the security agent of a given wrapper to the local requirements (full autonomy, middle autonomy or no autonomy), this agent checks or controls any access (from the global level to the local level) [3].

3. The Transaction Model in Federated databases

3.1. Conventional Transactions

Definition 1: Transaction is a means by which an application programmer can package together a sequence of database operations so that the database system can provide a number of guarantees, known as the **ACID** properties of a transaction. When the operations making up a transaction consist of both reads (that is, Selects) and updates, they represent an attempt by the application programmer to perform a consistent change of state in the data; when the operations of a transaction consist only of reads, they represent an attempt at a consistent view of the data [4].

A transaction Ti is a partial order with an ordering relation $\prec i$ where:

1. Ti = {ri (x), wi (x) / x is an object in the database} U { ai, ci }
2. An abort (ai) is inTi iff a commit (ci) is not in Ti.
3. A commit ci or abort ai is the final operation of Ti.
4. If ri (x), wi (x) $\in$ Ti, then either ri (x)$\prec$i wi (x) or wi (x)$\prec$i ri (x).

- Read(X) - which transfers the data item X from the database to a local buffer belonging to the transaction that executed the read operation.
- Write(x) - which transfers the data item X from the local buffer of the transaction that executed the write back to the database.
- Abort- Terminate and a transaction. Undo all transactions effects in the database.
- Commit- Make all changes that a transaction made permanent in the database and remove the transaction from the system.

3.2. Federated Transaction Model

In this section we can see the difference between federated database and other database systems regarding to large scale database; and also we compare the Conventional ACID properties with our transaction model ACID properties.

(1) Federated Database: consists of component database (>1) that are autonomous, yet participate in a federation to allow controlled sharing of their data. Association autonomy implies that the component database management systems (CDMS's) have full control over the data they manage. There is no centralized control in federated DBMS. The individual DBMS's cooperate to allow different degrees of integration. Applications and users may access data in the component databases either through the federated database or directly through the local DBMS.

(2) Multi-database: A multi-database system supports operations on multiple components DBMS's. This a broader classification since it makes no assumptions about autonomy. Federated database are a type of multi-database.

(3) Metadata base- This can be conceptually thought of as the “Database of Database” or a “Knowledge base” about databases. It basically contains information about ”information sources, content, and business rules”; across an enterprise in order to facilitate information integration.

(4) Inter operable system: These are the most loosely coupled information sharing systems. Global function is limited to simple data exchange and does not support full database functionally. Standard protocols are defined for communications among the nodes. Because the global system is not database-oriented, local systems may include other types of information repositories, such as expert systems or knowledge-based systems.

(5) Component Database: we will use a very broad and inclusive definition of this term to include any input or output source for the federated system. This includes conventional database systems (such as relational, object-oriented, hierarchical, network); files and file systems; object stores and data streams.

Table 1. Probability of all Databases being Available by Type & Number

Database Classification	Database Availability (individual)	Database Unavailability (min/year)	Number of Databases		
			100	1,000	10,000
Managed	99.00%	5256	0.3660	0.0000	0.0000
Well Managed	99.90%	526	0.9048	0.3677	0.0000
Fault Tolerant	99.99%	53	0.9900	0.9048	0.3679

On the other hand, based on the existing scalability problems with federated databases we proposed federated database schemas do not scale, because as the number of databases grows large the effects of component availability and failure become magnified. A serious problem with large federated databases is that the duration of the transactions also increases. Issues related to locking and deadlock become more problematic due to transaction duration, component failure and the availability of components. Using an availability classification from Gray & Reuter [5], we can analyze the result by taking (100, 1000, and 10,000) number of databases.

Note that as the number of databases grows, the probability of all them actually being available rapidly approaches zero due to the exponential nature of the function.

Pxyz = (Pind)n, where, 0≤ Pind<1 and Pind is the probability of each individual component being operational, n is the number of databases and Pxyz is the probability that all of the databases are operational. This is true even if we are dealing exclusively with fault tolerant machines. Furthermore these schemas have the undesirable property that they are only as reliable as the most unreliable participant. This is also a common criticism of many distributed database Computing schemes.

For large number of federated database we are design the following transaction model, which are its ACID properties different from the conventional transaction one.

A federated database F = {D,T}, where D = {D1,D2,D3,…,Dn} or a set of Component Database and T = {T1, T2, T3… Tn} or transactions. A federated transaction Ti = {Si1, Si2, Si3,…, Sim}U {ai,ci/ai ∈ Ti iff ci ∉ Ti}, where Sij is sub-actions, ai is abort and ci is commit.

In this model we also use local transaction Lk and compensating transaction Ctk in a component database Dk. (for example a long-duration transaction Ti representing a travel reservation which has three sub-transactions: Ti1 which makes airline reservations; Ti2 which reserves rental cars; and Ti3 which reserves a hotel room. Suppose that the hotel cancels the reservation. Instead of undoing all of Ti, we compensate for the failure of Ti3 by deleting the old hotel reservation and making a new one).

4. Conclusion

Database federations are still challenging since problems concerning distribution, heterogeneity and autonomy of component database systems joining a federation have not been fully solved yet. In this paper in general, an architecture, transaction model and frame work for federated database systems to support large numbers of member databases was presented. We concentrated on the transaction models of federated database system & distributed object kernel (DOK), in addition to different methods to access data and security approach that it is based on computational issues of aggregation constraints.

References

[1] Amit P.Sheth and James A.Larson. Federated Database Sysems for Managing distributed, heterogeneous, and autonomous databases. ACM Computing Surveys, 22(3): 183-236, September 1990.

[2] Zahir Tari, Omran Bukhres, John Stokes,etc. The Re-engineering of Relational Databases based on Key and Data Correlations.

[3] Zahir Tari and George Fernandez. Security Enforcement in the DOK Federated Database System.

[4] Patrick O'Neil and Elizabeth O'Neil. Database Principles, Programming, and Performance; 2nd edition, 2002.

[5] Jim Gray & Andress Router. Transaction Processing: Concepts and Techniques. Morgan Kaufmann, 1993.

基于 DOK 的联邦数据库中的事务模型

TEREFE Muleta　李瑞轩

（华中科技大学计算机科学与技术学院　武汉　430074）

摘　要　本文给出了构建联邦数据库系统的分布式对象核心（DOK）的体系结构和框架，在该系统中建立了分布式控制和决策支持，用于最大限度地实现并行性和可扩充性，通过时间戳和事务级的方法在分布异构系统中实现事务的并发控制。

关键词　事务管理，联邦数据库，死锁，对象核心，包装器

中图法分类号　TP311.13