TECHNIQUES FOR DATA REPLICATION ON DISTRIBUTED DATABASES

Adrian Runceanu, lecturer, *University Constantin Brâncuşi Târgu-Jiu*
Marian Popescu, lecturer, *University Constantin Brâncuşi Târgu-Jiu*
Mihaela Runceanu, teacher, *Ecaterina Teodoroiu High College Târgu-Jiu*

ABSTRACT.
Designing databases in a distributed environment is significantly more complex than designing databases in a centralized environment, largely because of the need to consider network sources, data partitions schemes, redundant data placement alternatives, an replication approaches. In this paper we present two alternatives design for database partitioning at both primary and replicate locations.

I. Introduction

Distributed computer environments are becoming the norm. In this type of environment, users share peripherals, data, and programs while the infrastructure provides system management across the network. Databases have been a part of this distribution evolution. There are defines two types of systems for linking distributed database environments: federated database systems and distributed database systems [9].

1. Distributed database management systems

By definition, distributed database systems should support the following functionality:

- Single logical database that is physically distributed.
- Single database definition, that is, schema.
- Global multiphase commit protocol across all resource managers that is completely system supported. This enables multiple source manager updates from within a single unit of work (a transaction).
- Global optimization of queries.
- Location transparency for all data access. This enables applications and/or users to modify and read data without knowledge of where the data is actually physically located.
- Coordinated recovery in the event of a system or disk failure at any site.

These features should allow support for all for types of distributed data access – remote request, remote unit of work, distributed unit of work, and distributed request.

2. Federated database management systems

By contrast, a federated database system is simply a collection of autonomous database systems (DBMSs). Each DBMS may support a different data schema, query language, and transaction management. The role of the federated database system software is to mask all these all these differences from any application process or user.

3. Data distribution alternatives

Because a distributed computing environment supports the sharing of peripherals, data, and processes, there is no underlying requirement that data be stored redundantly.

Therefore, data distribution alternatives include options, storage without data redundancy and storage with redundancy. Options where no redundancy is allowed include centralized and fragmented models. Options where data is stored redundantly include variations of a master/slave model or the update-any-where model.

Options with no data redundancy

When no redundancy is allowed, only a single copy of the data exists. Two distribution options are possible for this alternative: centralized and fragmented models.

Centralized model

The centralized model involves no data distribution. All data is stored in a single centralized resource manager. The use of the data can be distributed via local and remote access, but the data itself resides in the one location (figure 1).

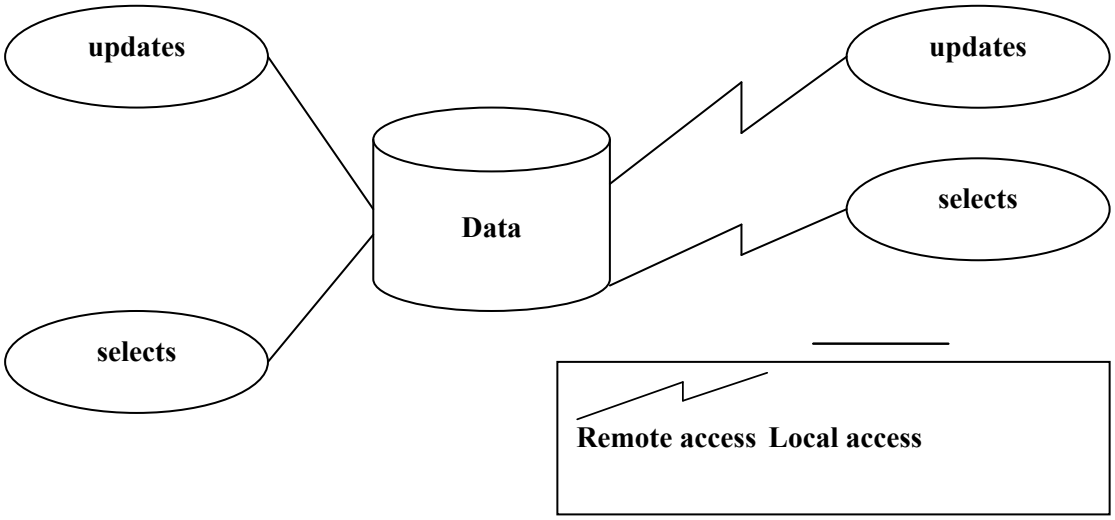

Figure 1. Centralized model

General characteristics of the centralized model include the following:

- Only a single copy of the data exists.
- Ownership of and ability to update the single data source are shared.

The main advantage to this lack of redundancy is that the data is always consistent and current. Because no redundancy exists, all users always see exactly the same data. In addition, the lack of distribution makes control, security, and maintenance very simple and straightforward. The biggest disadvantage of this model is that is a single point of failure. If a failure occurs and the data becomes unavailable, all systems stop functioning. A further concern is the potential for high network usage, which could be costly if remote access is high.

Fragmented model

This option involves the distribution of the single centralized data source identified in the centralized model. The distribution is accomplished via horizontal and/or vertical partitioning.

However, no data is stored redundantly with the exception of primary keys if vertical partitioning is implementing.

In the relational model, horizontal partitioning divides a table by rows. The basis for the division is usually the value in one or more columns within that table. For example, in a

banking firm, the partitioning column might be branch code with each branch "owning" and storing the data associated with the customers it serves.

Vertical partitioning divides a relational table by columns. Continuing with the banking example, vertical partitioning of a costumer table could have customer address information stored at the regional center and customer loan information stored at the branch that serviced that customer's loan request. Here the customer identifier column (primary key) would be stored in both the regional customer address table and the local customer table. As with the centralized model, the use of the data can be distributed via local and remote access, but only one copy of the data exists; it is partitioned across multiple distributed locations (figure 2).

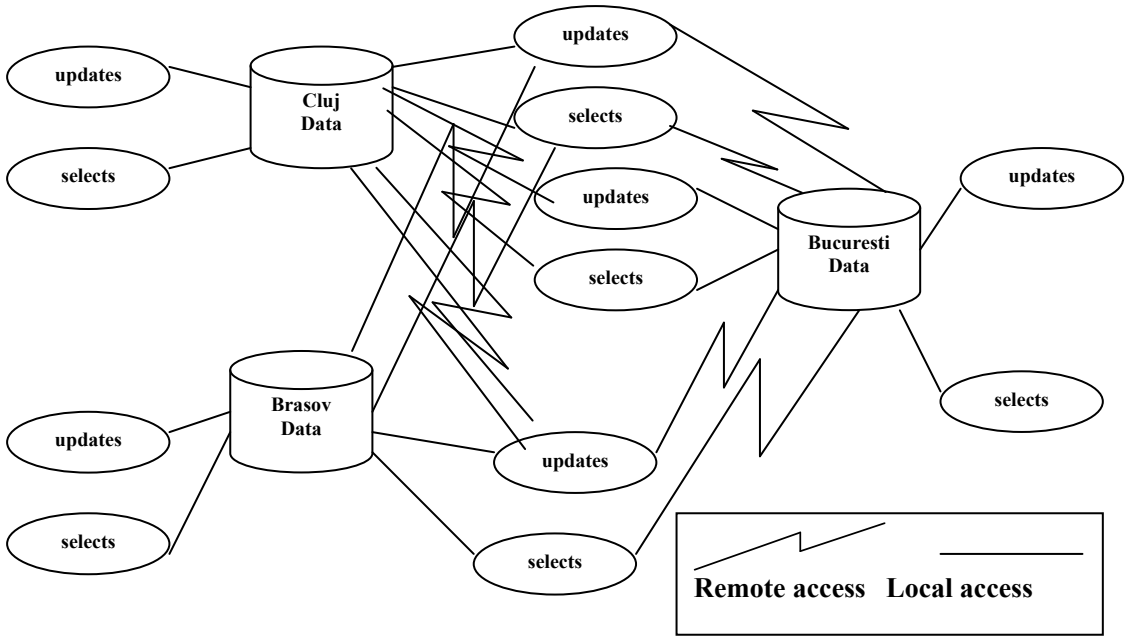

Figure 2. Fragmented model

General characteristics of the fragmented model include the following:

- Only a single copy of the data exists, but data is fragmented (partitioned) across multiple locations.
- Ownership of and ability to update the single distributed source are shared.

As with the centralized model, the biggest advantage is that data is always consistent and current. Redundancy exists only where vertical partitioning has been implemented and primary keys are carried redundantly. If this is the case, some sort of reconciliation should be implemented to ensure the integrity across the vertical partitions. In this model, control, security, and maintenance are slightly more complex than in the centralized model because multiple locations are used for persistent storage. The issue of a single point of failure has been reduced but not eliminated.

For example, the local branch, as identified in the above example, could complete a loan transaction even if the regional center was unavailable. The validation of the customer address information could occur later if this acceptable too business users. Network usage is reduced with respect to the centralized model. However, remote access is still required.

II. Use of database partitioning

1. Database partitioning at primary source

Database partitioning at primary source has the greatest potential benefit when a log pull mechanism of capture is used. The log pull mechanism is continually running process that extracts committed transactional information from the database log and sends it to the distribution mechanism of the asynchronous replication service. When this partitioning approach is used with a relational DMBS, the tables of a database are separated into groups. One group contains those tables or data that will definitely not be part of the replication process. This partitioning is performed to reduce the amount of log data pull mechanism has to scan. This is useful technique when a great deal of update activity occurs within the database (a large log) and only a very few tables are flagged for replication.

There are many disadvantages associated with this design alternative:

- *Partitioning within a database is complex.* Tables should not be separated so that referential integrity constraints cross databases. In other words, a primary key and all of its foreign key references should always be within the same database. This means that the database itself manages the defined referential integrity constraints. A further complication of poor partitioning scheme is potential for changing a local or remote unit of work into a distributed unit of work. This occurs when a transaction updates only within a single database pre-partitioning (remote unit of work), but updates across multiple databases after the partitioning (distributed unit of work).
- *Database administration and recovery become more complex*. Recovering two databases with separates logs to the exact same point in time is much more complex than just recovering one.
- *Database security is more complex*. Issues include how to handle data views that cross database, who owns the database view, where the user IDs reside, and how to handle a database stored procedure resides in one database but much access data in multiple databases.
- *Some flexibility is lost*. A relational table or piece of data may not be currently flagged for replication, but that does not mean that a demand for replica will not occur in the future.

This technique is not recommending for design the replication in distributed databases.

2. Database partitioning at the target replica

Database partitioning at target replica is also a design alternative. Partitioning at target replicas will have an impact on database recovery scenarios and on the degree of data access transparency for application code. The trade-offs here reflect the replication and database administrator perspective versus the application developer perspective. For administrators, if the target replica is partitioned so that every primary source replicates into its own target database, recovery and data reconciliation are simpler to handle. For application developer, data access is more complex with the addition of each new database.

The issues that should be considered in this design decision include the following:

- *Number of primary sources supplying data to the target replica*.
- *Number of tables per database*. Some data servers allow recovery only at database level. If the service-level agreement for database recovery demands a short allowable outage, then the more tables that reside within the database, the longer the amount of time for the recovery process.

- *Number of replication connections used to update the target*. Some vendors' asynchronous replication products provide only one connection per database for updating replication process. If multiple databases are used, then multiple connections can exists. This can increase throughput; however, be warned that the data stream flowing across each connection should be totally orthogonal. Orthogonal means that the data streams do not modify the same data. In other words, each data stream modifies only its designated portion of the data.
- *Characteristics of the recovery scenarios performed*. Orthogonal databases can be recovered independently of each other. In addition, if multiple databases are used and recovery is at a database level, recovery time is reduced. However, having multiple databases also makes database recovery to a point in time very difficult because synchronization within multiple logs to the very same moment is not easy.

III. Implementation

Replicated data are becoming more and more of interest lately. The use of data replication has many advantages including the increased read availability (many operations can be handled locally, reducing communication costs and delays) and reliability (if one site is down, or has lost some of its data, the data is likely to be available at another node) but makes the data updating more complicated.

While data copying can provide users with local and much quicker access to data, the problem is to provide these copies to users so that the overall systems operate with the same integrity and management capacity that is available within a centralized model. Managing a replicated data is significantly more complicated than running against a single location database. It deals with all of the design and implementation issues of a single location and additionally with complexity of distribution, network latency and remote administration [7].

In view of the above, the new algorithms ensuring the best possible processing parameters (e.g. data consistency, failures resilience, query execution time) of replicated data are being searched for. There are many different approaches to replication, each well suited to solve certain classes of problems. Various ways of supporting replicated data (identical copies, primary/secondary, snapshot) as well as various strategies of its updating are taken into consideration. The known update strategies can be classified as follows [3]:

- Synchronous all,
- Synchronous available,
- Quorum-based,
- Primary/secondary,
- Primary/backup,
- Independent.

Various update methods, based on these strategies, have been developed e.g. *Read One Write All, Directory-Oriented Available Copies, Quorum Consensus, Tree Quorum Protocol, Triangular Grid Protocol, Multi-Airline Reservation, Independent, Remote Backup Procedure, As Soon As Possible, Copy Token,* etc. [2][3][4][5][7]. However, the problem of managing replicated data is still current.

IV. The Experimental Distributed Database Management System

The research presently carried out is the continuation of the studies referring to the already created experimental distributed database management system (EDDBMS) [8]. The system is assumed to run on a cluster of workstations connected by the Ethernet local network. The main designed and implemented component is *application processor.* This

module is responsible for coordination of the access to data items located on various nodes and managed by the chosen, as *data processors*, database management systems (Ingres, Postgres95, MS SQL Server).
There are the following elements of the application processor: *clients* and *local servers*. A client constitutes an interface for an end user. It is responsible for making the syntactical and semantic analysis of user queries and their decomposition into a set of sub-queries operating on physical data items. The global schema used by the client is stored in the repository being a centralized local database. Through an additional locking mechanism independent of the locks used by data processors, the client module controls the concurrent access to the data, including global deadlock detection and resolution. It also sends the sub-queries to the local servers, manages the distributed transactions and presents their results.

A client, managing global transactions, cooperates with a local server, which performs concurrent sub-transactions with participation of the local DBMS. The cooperation between a local server and a data processor is achieved owing to the embedding SQL in C language and the externalization of the transaction commitment protocol.

Any number of client processes and one local server process for each data processor used can simultaneously run in the system. Independent processes running on the network communicate through virtual shared memory (VSM) called *tuple space* (TS), passing data units called *tuples*. The implementation is based on the distributed computing environment *Paradise.*

The system, implemented in this way, is used to form an environment for developing and testing algorithms of the selected DDBMS mechanisms: query processing, concurrency control, distributed transaction commitment, and maintaining data replication.

References

[1] P.A. Bernstein, V. Hadzilacos, N. Goodman: *Concurrency Control and Recovery in Database Systems,* Addison–Wesley, 1987.
[2] G. Coulouris, J. Dollimore, T. Kindberg: *Disributed Systems, Concepts and Design,* Addison–Wesley, 1994.
[3] S. Ceri, M.A.W. Houtsma, A.M. Keller, P. Samarati: *A Classification of Update Methods for Replicated Databases,* via Internet, May 5, 1994.
[4] D. Agrawal, A.El. Abbadi: *The Tree Quorum Protocol: An Efficient Approach for Managing Replicated Data*. in Proc. of VLDB Conf. pp 243-254, 1990.
[5] C. Cho, J. Wang: *Triangular Grid Protocol: An Efficient Scheme for Replica Control with Uniform Access Quorums.* Theoretical Computer Science, Volume 196, Numbers 1-2, Apr 6,1998.
[6] S. Ceri, M.A.W. Houtsma, A.M. Keller, P. Samarati: *Achieving Incremental Consistency among Autonomous Replicated Databases.* in Proc. DS-5, Semantic Interoperability, Lourne, Australia Nov 1992.
[7] G. Schussel: *Replication, The Next Generation of Distributed Database Technology,* via Internet: http://www.dci.com/speakers/archive/replica.html
[8] P. Bajerski, M. Choplek, K. Harezlak, H Josinski : *Environment for Distributed Database Management Algorithms Evaluation Based on Virtual Shared Memory*. Proceedings of the 16th IASTED International Conference APPLIED INFORMATICS, held February 23–25, 1998, Garmisch–Partenkirchen, Germany.
[9] Marie Buretta: *Data Replication. Tools and Techniques for Managing Distributed Information*, Wiley Computer Publishing, 1997.