THE DISTRIBUTED TRANSACTION MANAGEMENT IN THE MODERN ECONOMY

NICOLETA MAGDALENA IACOB (CIOBANU)[1]
UNIVERSITY OF PITESTI, DEPARTMENT OF COMPUTER SCIENCE,
1 TARGU DIN VALE STREET, PITESTI, ARGES, ROMANIA. 110040
nicoleta.iacob_2007@yahoo.com

Abstract:
The present society is without any doubt an informational society. The implementation of complex applications, created to comply with the business needs of various organizations - many of them with geographically distributed offices - led to the development of new computer networks capable to assure the rapid transmission of information.
This paper describes the types of distributed transactions with their properties and operations. The aim of distributed transactions is to ensure concurrency and data consistency for all users of the database. In addition, we analyze ways of managing transactions.

Key words*: distributed database, transactions, two-phase commit protocol - 2PC, concurrency control.*

JEL classification: *C88, L21.*

1. Introduction

In the current business globalization context, the organizational environment of a company must adapt to the competitive market. Economic growth of a company depends crucially on its ability to integrate, customize and expand software applications, in a flexible mode, in order to provide instant and interactive access to all users to company resources.

The distributed system is designated to be applied especially on organizational structures of companies which are distributed geographically. For organizations that are expanding globally, the exchange of data between databases and multiple applications has become very important.

A **distributed database** (DB) is a set of databases located on several computers that is viewed by the application as a single database (located on a single computer) - database is fragmented and divided on multiple servers from different nodes on the network. Every DB must at least support the ability to "create" new data content (store new data values in the DB) and the ability to retrieve existing data content (Rahimi, Haug, 2010).

A **distributed database management system** (DDBMS) is a set of programs that allows the management of distributed database and makes the distribution transparent to the users. The objective of transparency is to make a distributed system appear similar to a centralized system (Iacob, 2010).

In a database distributed on nodes of a network, access to many data objects is usually done through transactions. A **transaction** allows the programmer to group one or more SQL (Structured Query Language) instructions into a single, indivisible one.

[1] This article was produced under the project: "Development of the doctoral schools by providing fellowships for young PhD students with frequency" - ID 52826.

The transaction begins with the first executed SQL statement and it ends when the effects of the transaction are saved or rolled back.

Transactions management ensure concurrency and consistent of data for all database users. In the absence of transaction management, data quality and accuracy would be compromised, and the database would become almost unusable.

Synchronization mechanisms of accesses created to maintain database integrity are called **concurrency control.**

2. Distributed transactions

In a multi-user database system the procedures that cause changes to a database or retrieve data from the database are called transactions (Beynon-Davies, 2004).

The transactions are atomic units of data update, by which a database goes from a coherent state into another coherent state.

So, a **transaction** is a sequence of actions, executed by the user, which maintains consistence and security of a database. Intuitively, transactions take data from the database, perform some calculations and change some values of the database. In a computer network is very important to not allow access to unauthorized persons to the information sent between computers (Defta, 2010).

For each transaction, Oracle program allows two alternatives. If the SQL statements in a transaction are finalized normally, the effects of the transaction become permanent in the database. This is called *completing (saving) the transaction.* If, however, from some reason, SQL statements are not completed successfully, the effects of the transaction are removed from the database and the transaction is finished. Removing the effects of a transaction is called *rolling back the transaction.*

Example transaction:

```
begin transaction
        extract (account1, amount);
        submit (account2, amount);
        updateHistory ();
commit transaction
```

If all operations succeed: COMMIT

If one operation fails: ROLLBACK

Running a successful transaction is not equivalent with finishing the transaction. A successful execution of a transaction means that SQL statements were executed without errors. However, the effects of successfully executed transactions may be rolled back as long as the transaction is not completed.

Since transactions that are executed in a distributed system can process data located in one place, or data located in separate systems, we distinguish two types: local transactions and global transactions. *Local transactions* have access to information contained in a single local database and can process this information. *Global transactions* have access to information located in multiple databases and can process this information.

2.1. Transaction properties

To ensure integrity of the data, we require that the database system maintain the following properties of the transactions:

Atomicity: The transactions are formed of several instructions grouped together in a "macro-instruction" that acts as an indivisible unit. In other words, if in a transaction we group some instructions then all those will be executed or not.

For example, employees payments made by an accounting program.

```
paid: = 0;
for i: = 1 to employees  do
begin
        Payment[i]  := Payment[i]  + wage[i];
        paid: = paid + wage[i]
end;
budget: = budget  - paid;
```

If we assume that the changed values are committed to a database, then in case of a software or hardware error occurred somewhere in the execution of "for" statement, some salaries will be paid, but the amount will not be deducted from the budget. The program cannot be resumed from the beginning, because some salaries have been paid, but others do not. The solution is to execute all these instructions in a single transaction, and if this is broken before completion, then the changes made will be rolled back.

Consistency: Taking into account all the variables of a program, they will each have a value at a given time. But not every combination of values is possible. This rule is called invariant, so always have to be true. In the example of above program an invariant is: "the amount paid and the current budget should have the same amount" (payments balance is zero).

Independence: This property refers to a context where multiple programs simultaneously access same set of data (concurrent activity). A program should not be able to access partial data of another program, because they would not be correct. To illustrate that we consider a program that implements a salary raise:

```
for i: = 1 to employees  do
wage [i] = wage [i] * 110/100; {10% raise}
```

What consequences could have the execution of this program simultaneously with the above? One of these consequences could be that this program will start execution after the previous one, somewhere to reach the same index and then to go forward. Thus some employees will receive higher wages (the latter), and others not. This would lead to an inconsistency in the payments balance of that company. It is possible to avoid this if all the fragments of program are included in a transaction.

Durability: Refers to the programming with persistent data structures. A simple program is executed each time the same way, having as the starting points the same initial values. But a program that operates with a database or a file has data structures that are on disk and can overcome an error generated by the program. If executing the salary program twice, the growth wage operation can have different results, because the second increase is applied over first, and the results of first execution are saved on disk. Persistence is very important in databases, where all operations are performed on permanent data.

These properties are often called the **ACID** properties; the acronym is derived from the first letter of each of the four properties (Silberschatz, Korth & Sudarshan, 2010) - **A**tomicity, **C**onsistency, **I**solation, **D**urability.

Compliance with ACID properties in the local transactions execution process implies participation of all components of the database management system that are involved in the management of transactions.

In the case of global transactions, maintaining ACID properties is significantly more complicated, having in mind that in the execution of a transaction take part multiple nodes of a network. Faults in one or more of the nodes, and also communication channels failures, will lead almost certainly to incorrect processing of information.

2.2. Transparency of transactions

Such transparency ensures that all transactions preserve integrity and consistency of distributed DDB. A distributed transaction accesses data stored in more than one location within the network. Each transaction is divided into sub-transactions, one for each visited site. Sub-transactions are executed in parallel on the sites and concurrently on each site. So, DDBMS has a special role in the transactions management. Related to transactions transparency, a special attention is given to transparency concurrency and errors. DDBMS provides *concurrency transparency* if the results of all concurrent transactions (distributed or not) are performed independently and are logically equivalent with results obtained if those transactions would have been executed in an arbitrary serial order.

Error transparency requires a recovery mechanism to assure that transactions are atomic in case of errors: either all operations of a transaction are performed or no operation is executed. Once a transaction has been executed, the transformations committed to the database are permanent.

2.3. Transaction management

Concurrent transaction management system includes two main modules:

- *Transaction control module* (TCM) which receives and order transactions, ensuring database consistency preservation.
- *Data management module* (DMM) which executes transactions (orders of reading, writing) and ends transactions (check orders, cancellation). DMM contains two modules:
  - recovery management module;
  - memory management module.

DMM types:

- DMM *aggressive* - the transactions are delayed as little as possible and are released into execution as quickly as possible, even if immediately after launch are canceled, due to a conflict.
- DMM *conservative* - the transactions are delayed as long as possible in order to analyze the read and written data for each transaction.

The application type determines the choice of relational DBMS as follows:

- DMM aggressive - if application contains transactions that rarely generate conflict.
- DMM conservative – if request is to prevent conflicts with the price of losing speed during further analysis of the transactions.

The interface commands with application programs of the transaction management module are:

1) **BEGIN TRANSACTION** marks the beginning of transaction execution. The administrator of transactions records in a table when receiving the order:
   - transaction identification number,
   - the application from which these transactions are generated,
   - transaction occurrence time, etc.
2) **READ** works as follows:
   - if the referenced access unit is stored locally, management module returns its value,
   - if the referenced access unit is not stored locally, it selects a copy of unit and requires its value.
3) **WRITE** determine writing the access units (locally).
4) **COMMIT** announce the successful completion of the transaction, validation of changes made to the database and visibility of changes made to other

transactions; from this point, committed changes cannot be canceled, or lost by a subsequent failure of the system (if the recovery mechanism of DBMS is working properly).

5) **ROLLBACK (or ABORT)** marks the failure of a transaction; any effect that it had on the database needs to be canceled (by "rolling back" of operations). There are some systems in which after ABORT command the management module will indirectly try to recover a part of the canceled transaction actions.
6) **UNDO:** is similar rollback operation, but applies to a single operation, not to an entire transaction.
7) **REDO**: specify that some operations of a transaction must be executed again in order to validate the entire transaction.
8) **END TRANSACTION** marks the end of execution for a transaction.

2.4. Two-phase commit protocol

The validation protocol in two phases *(two-phase commit)* - 2PC relates to a transaction that makes the database upgrade.

In the first phase:

The source (a database server) will require targets (other database systems) to prepare for executing the transaction and will wait for confirmation from them.

When all targets are prepared, the source will ask the execution of transaction and will wait for each target to report the success.

Each server responds with:

- The READY message, if it is prepared to validate the transaction; after this point, the server cannot cancel transaction unless the coordinator asks.
- The REFUSE message, if it cannot validate the transaction.

Phase two:

If coordinator received READY from all servers - all targets were successfully committed the transaction - it send COMMIT message to all servers and each server will validate the transaction.

If coordinator received REFUSE from at least one server, which means that at least one target cannot commit transaction, then it sends ROLLBACK to all servers and each server will cancel the transaction.

If all servers are working properly and there are no network errors, 2PC ensure atomic distributed validation for transactions.

However, taking into account the possibility of errors occurrence, other protocols are necessary to be used during validation process.

2.5. Concurrency control

If the transactions are serialized, time is wasted when the server waits for an input/output operation, or during other breaks. On the other hand, if the transactions are executed in parallel, we have concurrent transactions. Concurrency will increase efficiency, but also necessitate control mechanisms to be implemented because otherwise it can lead to database inconsistency.

Concurrency control in a database is the activity of coordinating transactions which operate in parallel, access shared data and can potentially interfere with each other. Concurrency control together with a recovery protocol guarantees the ACID properties of a transaction. The goal of concurrency control is to keep transactions atomic.

Concurrency control problem arises for the operation of reading (it is possible to allow multiple transactions to read data from a common data entity) and the operation of writing (do not allow many transactions to write the same data entity).

There are two large categories of concurrency control:

- **Optimistic concurrency** - Delay in synchronization between transactions until the operations are completed. In this case, the conflicts are unlikely to happen but are visible only at the end of the operation, making the rollback operation more expensive.
- **Pessimistic concurrency** - potential concurrent execution of transactions is synchronized during the execution cycle. The locks are more probable but, because they are visible, the cost of the rollback operation is minimized.

Distributed transactions provide consistent execution units which group different operations in different nodes and keep the database consistent. *Two-phase commit* protocol 2PC ensures transaction completeness and consistency. For large distributed transactions involving different nodes there is the solution of decomposing these transactions in smaller units.

3. Conclusions

The transactions enable Oracle database user, application programmer and database administrator to ensure data consistency. Data *coherence* provides the user a coherent image of data, which consists of data stored by other users as well as changes made by the user himself. Data *concurrency* allows users to access data that are used by many other users simultaneously. In the absence of transactions to coordinate simultaneous access and data consistency, the user would gain incorrect data values, lost updates and reading failures.

In the new economy and society based on knowledge, the main factors which contribute to economic growth and development are knowledge, innovation and technical progress. Following the demands of increasing availability, reliability and flexibility of database systems and also technology and communications progress, distributed database systems are now widespread.

The process of creating a distributed database systems is justified when performing with large database systems, which provide support for critical business activities, when the organizations expect frequent expansions or when changing the existing infrastructure to maximize efficiency by interconnecting a set of existing centralized databases.

REFERENCES

1. Beynon-Davies, P. (2004). *Database systems*. (3rd ed.). New York: Palgrave-Macmillan.
2. Defta, L. (2010). "Network security attacks. ARP Poisoning case study.", *Journal "Constantin Brâncuşi" University of Târgu Jiu Annals* - Economics Series, No. 4/2010, pp. 174-181.
3. Iacob, N. (2010). "Data replication in distributed environments", *Journal "Constantin Brâncuşi" University of Târgu Jiu Annals* - Economics Series, No. 4/2010, pp. 193-202.
4. Rahimi, S.K.; Haug, F.S. (2010). *Distributed database management systems*. A Practical Approach, IEEE, Computer society, New Jersey: John Wiley & Sons, INC.
5. Silberschatz, A.; Korth, H.F.; Sudarshan, S. (2010). *Database System Concepts*. (6th ed.). McGraw-Hill.