UDC 004.031

Włodzimierz Khadzynov, Mateusz Maksymiuk
Politechnika Koszalińska, Katedra Inżynierii Komputerowej
ul. Śniadeckich, 2, 75-453 Koszalin, Polska
e-mail: hadginov@ie.tu.koszalin.pl, mateuszmaksymiuk@wp.pl

Model and classification of database replication techniques

Construction of a universal mathematic-logical model of replication processes in data bases is considered. The fundamental definitions of notions for different types of replication are given. The classification of replication techniques is proposed and the respective efficiency criteria are determined. The variants of realization for system architecture, locking strategy, servers interaction, transaction termination, database platforms, correctness criteria of transaction terminator are discussed.

Key words: *database replication, distributed database, transaction, locking transactions, isolation transactions.*

1. Introduction

Recently there was occurred rapid development in a database replication area. Many new methods and various techniques were proposed. Because database replication is used in many distinct scenarios (i.e. security, efficiency, accessibility), there is a necessity for different techniques which are adapted to given requirements. Any implementation of database replication technique must be based on some fundamental analytic model which describes the behaviors of the system at high level of abstraction.

Model of the database replication process is the first stage during development of new replication technique. Today there are known many various models of database replication. Although every algorithm of database replication is unique and different, it can be described using certain universal model and classified by standard criteria. In this paper is presented the detailed classification of database replication issue, based on an elementary criterion which allows to divide various techniques into different classes. Moreover there is included a brief description of each case. The main purpose of this work is a presentation of general analytic model of database replication, its classification and various aspects of its techniques, methods and implementations.

2. Model and definitions

2.1. Distributed database

We consider a system S which consists of a finite set of nodes with its cardinality equal to n — formula 1:

$$S = \{s_1, s_2, \ldots, s_n\}. \tag{1}$$

The term *node* (also known as *site*) means a single database which belongs to a distributed database. All nodes are in touch with one another through the network. In this paper we omit the influence of the network topology to a distributed system and we suppose that all nodes are connected through the WAN network in the way: peer-to-peer.

Each node s of a considered system at any time can be in one of two states (correct or faulty), which is returned by function $State(s)$ — formula 2:

$$State(s) \in \{correct,\ faulty\}. \tag{2}$$

Moreover we suppose that when a node is in faulty state, it behaves as it does not exist, which means it does not response to any request/messages. It means that we exclude *Byzantine failures*, in which server may give false responses.

Distributed database Ω, which represents nodes of system S, contains a finite set of data items x_j (e.g. records) — formula 3:

$$\Omega = \{x_1, x_2, \ldots, x_k\}. \tag{3}$$

Distributed database Ω is partitioned on partitions D_l, where $l \in \{1,...,n\}$. Each partition D_l is replicated at m additional nodes. The case when $m = 0$ implies replication does not exist, the case when $m = n-1$ implies full replication, for intermediate values: $m \in \{1, n-2\}$ there is a partial replication. In our consideration there is assumed a uniform partitioning, it means each partition is replicated at identical number of nodes.

For each data item x_j exists at least one node, which contains x_j — formula 4:

$$\underset{x\in\Omega}{\forall}\ \underset{s\in S}{\exists}\ x \in Items(s). \tag{4}$$

For each node $s \in S$ function $Items(s)$ returns a set of all data items, which are stored at node s. Function $Sites(x)$ returns a set of all nodes which contain a data item x.

2.2. Distributed transactions

Transaction t is a finite sequence of read/write operations o_i — formula 5. Every transaction is terminated by *commit* or *rollback* action:

$$t = (o_1, o_2, \ldots, o_m)\,, \tag{5}$$

$$t \in T\,. \tag{6}$$

Every transaction t belongs to a set of all possible transactions T, which can be executed on distributed database — formula 6.

Functions in formulas 7 and 8 return sets of data items that are read and written by transaction t.

$$RS(t)\,, \tag{7}$$

$$WS(t)\,. \tag{8}$$

Current state of transaction t is returned by function $State(t)$ presented in formula 9. This state can have one of the four values: *Executing* (transaction executes read/write operations), *Certifying* (verification of transaction before commit), *Aborted* (transaction was rolled back), *Committed* (transaction was committed).

$$State(t) \in \{Executing, Certyfying, Aborted, Commited\}\,. \tag{9}$$

Distributed transactions can be divided in two groups (classes) according to classification proposed in [1]. Transaction of class I, means such transaction $t \in T$ started at node $s \in S$, which operates only on data items that belong to that node (s) — formula 10. For each transaction $t \in T$, function *Beginned At*(t) returns the node in which given transaction was started.

$$t \in T : \exists_{s \in S} \begin{bmatrix} Beginned\ At(t) = s \\ \wedge \\ [RS(t) \cup WS(t)] \in Items(s) \end{bmatrix} \Leftrightarrow t = t^{CLASS\ I}\,. \tag{10}$$

Transaction of class II, means such transaction $t \in T$, started at node $s \in S$, which operated on data items that do not belong to local node s — formula 11.

$$t \in T : \exists_{\substack{s_0 \in S \\ s_1 \in S \\ s_0 \neq s_1}} \begin{bmatrix} Beginned\ At(t) = s_0 \\ \wedge \\ [RS(t) \cup WS(t)] \notin Items(s_0) \\ \wedge \\ [RS(t) \cup WS(t)] \in Items(s_1) \end{bmatrix} \Leftrightarrow t = t^{CLASS\ II}\,. \tag{11}$$

Transactions of class II are more difficult to execute, because they need to reference to remote nodes, i.e. during pessimistic locking, which reduces system efficiency. This aspect was discussed in detail in [1].

3. Classification

In order to classify replication issue, first there must be defined appropriate criteria, according to which such classification occurs. Because researches on database replication issue are conducted for some time, there was proposed several classification schemas and criteria. One of the most known classifications was proposed by Grey in [2]. It is based on two basic criteria: where and when updates take place. More detailed classification presented by Wiesmann in [3, 8] is based on three parameters: system architecture, interaction between servers and transaction termination method. In this paragraph is presented classification of replication based on both proposals extended with additional criteria.

3.1. Transaction propagation mode

One of the basic criteria of the notion of replication is related to time domain. This criterion is based on delay in time at transaction execution at different nodes of distributed database. There are two possible cases: synchronous (immediately execution) and asynchronous (deferred execution).

3.1.1. Synchronous mode

In synchronous mode transactions updating data, are executed at all nodes of a system simultaneously ensuring full consistency and integrity of data. In any nodes $s_1, s_2 \in S$, for any data item $x \in \Omega$ which belongs to both nodes, at any time $t \in <0, \infty>$, the value of this data item is identical — formula 12 (we consider only committed transaction). Function $Value(s, x)$ returns current value of data item $x \in \Omega$ at node $s \in S$.

$$\underset{\substack{(s_1,s_2)\in S \\ State(s_1)=Correct \\ State(s_2)=Correct}}{\forall} \quad \underset{\substack{x\in\Omega \\ x\in Items(s_1) \\ x\in Items(s_2)}}{\forall} \quad \underset{t\in<0,\, \infty>}{\forall} \; Value(s_1, x, t) = Value(s_2, x, t) . \tag{12}$$

In this mode distributed database meets 1-*copy serializability* condition. It means that result of execution of any sequence of transactions at distributed database is identical as the result of execution of the same sequence of transactions on the single database. Thus any end-user cannot recognize that he is working with distributed database but he is convinced to be working with single database. Synchronous mode causes many problems with its efficiency described at [6].

3.1.2. Asynchronous mode

In asynchronous mode transactions are executed only at local nodes. Propagation of transactions to other nodes is realized in deferred times. It means that for every two nodes $s_1, s_2 \in S$ for any data item $x \in \Omega$ which is contained in these two nodes, for an arbitrary transaction $t \in T$, which modified data item x and was committed at time t_0 at node s_1, there exists such pair of finite times $(t_1, t_2) : t_1 \prec t_2$, that values of data item x will be different at time t_1 and identical at time t_2 — formula 13.

$$\underset{\substack{(s_1,s_2)\in S\\ State(s_1)=Correct\\ State(s_2)=Correct}}{\forall}\quad \underset{\substack{x\in\Omega\\ x\in Items(s_1)\\ x\in Items(s_2)}}{\forall}\quad \underset{\substack{t\in T\\ t:(s,t_0)}}{\forall}\quad \underset{\substack{(t_1,t_2)\\ t_1\prec t_2}}{\exists}\begin{bmatrix} Value(s_1,x,t_0)\neq Value(s_2,x,t_0+t_1)\\ \wedge\\ Value(s_1,x,t_0)=Value(s_2,x,t_0+t_2)\end{bmatrix}. \qquad (13)$$

Minimum value of time t_2 is called propagation time of modifications to all nodes of distributed database. The less value of propagation time is the behavior of distributed database is near synchronous mode. Because of deferred updates the transactions are executed much faster, but its disadvantage is temporary data inconsistency.

3.2. System architecture

Architecture of distributed database system is an elementary criterion of classification of replication systems. There can be distinguished two cases: centralized (*Primary Copy*) in which exist some special nodes and decentralized (*Update Everywhere*) where all nodes are uniform.

3.2.1. Primary Copy

In *Primary Copy* [6, 7] approach exists one special node in each partition of data items, which is called *Master*. The *Master* node is responsible for execution of transactions which modify data (or sets up the order of transaction processing) and back propagation of these results to *Slave* nodes.

The *Slave* nodes are all nodes besides *Master* nodes. At *Slave* node there can only occur read of data and any updates of data must be redirected to appropriate *Master* node (Fig. 1). Unfortunately such solution brings crucial disadvantages. First, the *Master* node may become «bottleneck» and cause low efficiency. Second, any failure of *Master* node stops the execution of all update transactions that operate on data items from given partition. One solution of that problem is the election of a new *Master* node in case of failure of the previous one. System efficiency could be improved by incrementing the number of partitions, which also increment the number of *Master* nodes and in case of single *Master* node failure only a small part of a system is disabled from updates.

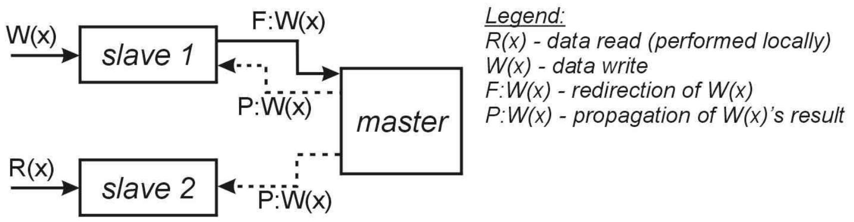
W(x)
slave 1
F:W(x)
P:W(x)
master
R(x)
slave 2
P:W(x)
Legend:
R(x) - data read (performed locally)
W(x) - data write
F:W(x) - redirection of W(x)
P:W(x) - propagation of W(x)'s result

Fig. 1. Diagram of Primary Copy model

3.2.2. Update everywhere

In *Update Everywhere* [6, 7] model the modification of data can occur at any node of distributed system. All nodes have equal ability to read and write the data, which ensures system symmetry. Each modification is propagated to all other nodes of given partition, however read-only transactions are executed locally at current node (if it contains requested data). This method is called ROWA (*Read One, Write All*). To ensure data consistency and integrity there are used appropriate group communications protocols combined with proper locking strategies. The main advantage of such system is symmetry and lack of special nodes (decentralization) what gives better reliability and eliminates «bottleneck» nodes (such as *Master* node). But on the other side, decentralization requires more sophisticated and complicated protocols, which cause much more network traffic during group communication. Unproper selection of these protocols and its parameters can greatly decrement system efficiency. Common model of *Update everywhere* is shown in Fig. 2.

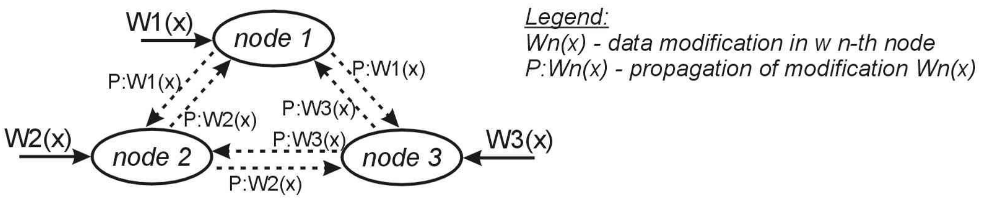
W1(x)
node 1
P:W1(x)
P:W1(x)
P:W2(x)
P:W3(x)
W2(x)
node 2
P:W3(x)
node 3
W3(x)
P:W2(x)
Legend:
Wn(x) - data modification in w n-th node
P:Wn(x) - propagation of modification Wn(x)

Fig. 2. Model of Update Everywhere replication

3.3. Locking strategy

Execution of transactions in distributed database must concern concurrent access of many other transactions to the same data. Because of that there are required specific algorithms which ensure secure access to data. These methods are used to eliminate occurrences of conflicts and guarantee serializable access to the database. Generally, locking strategies could be divided into three classes: pessimistic, optimistic and semi-pessimistic. Detailed analysis of influence of locking strategies to replication efficiency is shown in [4].

3.3.1. Pessimistic locking

Pessimistic locking strategy assumes resource locking during transaction execution (Fig. 3). Each read or write of data causes to acquire appropriate lock. Conditions required to acquisition of specific locks are shown in formulas 14 (read lock — shared) and 15 (write lock — exclusive):

$$[LockWrite(s_1, x) = ok] \Leftrightarrow \underset{s \in Sites(x)}{\forall} \begin{bmatrix} Is\,\mathrm{Re}\,adLocked(s, x) = false \\ \wedge \\ IsWriteLocked(s, x) = false \end{bmatrix}, \tag{14}$$

$$[Lock\,\mathrm{Re}\,ad(s_1, x) = ok] \Leftrightarrow \underset{s \in Sites(x)}{\forall} [IsWriteLocked(s, x) = false]. \tag{15}$$

Besides the type of lock (read or write), important thing is the kind of locked resource. There can be locked various types of objects, such as single records, set of records, tables, schemas etc. For example two transactions would not cause a conflict, if they operate on the same table but separate sets of records. Locking eliminates possibility of conflicts occurrence, but is not an ideal method because it reduces distributed database efficiency and may cause deadlocks.

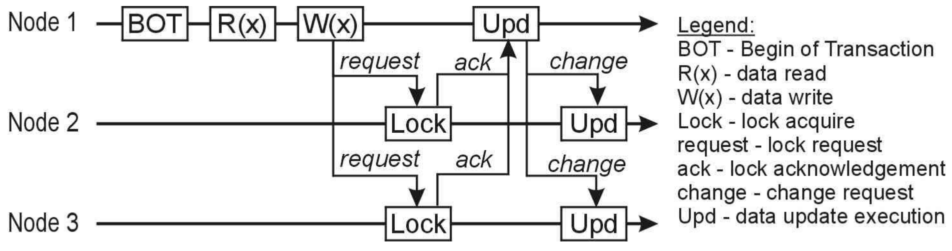

Fig. 3. 2-Phase Locking (2PL) protocol

3.3.2. Optimistic locking

Unlike pessimistic locking, the optimistic strategy does not block any resources during transaction execution. Thus there is no possibility of deadlocks occurrence. Because there is no locking (which requires additional communication interaction between nodes), there is no need for usage of locking protocols what leads to less network traffic and better efficiency. Optimistic method allows concurrent transactions to continue simultaneous execution. Verification of transaction occurs just in its second phase, during commit when occurrence of any conflicts is checked. In case of failure (when conflicts occurred), what means that current transaction conflicts with these which have been already verified, current transaction is aborted. Condition for transaction t_1 abortion during its verification is shown in formula 16. This is a main disadvantage of optimistic locking strategy: some part of transactions is aborted or alternatively requires repeated execution:

$$Aborted(t_1), t_1 \in T \equiv \begin{bmatrix} State(t_1) = commiting \\ \wedge \\ \exists t_2 \in T, \neg(t_1 \to t_2) \\ \wedge \\ \begin{pmatrix} RS(t_1) \cap WS(t_2) \neq \varnothing \\ \vee \\ RS(t_2) \cap WS(t_1) \neq \varnothing \end{pmatrix} \end{bmatrix}. \tag{16}$$

Above condition contains several elements. The first one requires that transaction has to be in a commit phase. Next is the time criterion. The symbol $(t_1 \to t_2)$ means that transaction t_1 precedes transaction t_2, which is equivalent that t_1 which started before start of t_2. Thus expression $\neg(t_1 \rightarrow t_2)$ means that t_2 began not later than t_1. The last two elements check conflict occurrence between two transactions t_1 and t_2.

3.3.3. Semi-optimistic locking

Semi-optimistic locking strategy does not bring any new algorithms, but is a combination of both previous methods. For example, for I class transactions the pessimistic locking is used and for II class transactions the optimistic one is used. Such solution gives reasonable compromise between efficiency and amount of aborted transactions.

3.4. Server interaction

The network interaction between servers (nodes) is very time-consuming. Furthermore many nodes may be located in a very long distance from each other. Thus very important issue is communication interaction between nodes. There are used many various protocols based on group communication to implement database replication. The main criterion of classification of these protocols is the number of messages transmitted during execution of a single transaction. This number is expressed by function $f(k)$, which argument k is the number of operations (insert, update, delete) in given transaction.

3.4.1. Constant interaction

In constant interaction case the number of messages $f(k)$ transmitted in a single transaction is constant, regardless of number of operations k contained in current transaction. This relationship is presented with big-*O* notation in formula 17.

$$f(k) = O(1). \tag{17}$$

In most cases all operations contained in transaction are transmitted in a single message. Because of this, a distributed system is less sensitive to the number of operations executed in a single transaction, what causes better efficiency.

3.4.2. Linear interaction

Unlike constant interaction, in the linear case the number of transmitted messages is variable. Amount of messages is directly proportional to the number of operations in a single transaction — formula 18:

$$f(k) = O(k). \tag{18}$$

Usually, single message contains only one database operation. Thus if a transaction has many operations, it requires transmitting many messages. Single message can contain SQL statement or record from transaction log.

3.5. Transaction termination method

Execution of distributed transactions requires proper co-ordination of all nodes of a distributed database. To maintain data consistency and integrity, there must be used appropriate techniques. Depending on the way of implementation we can distinguish two groups of algorithms: *voting techniques* and *non-voting techniques.*

3.5.1. Voting techniques

Voting techniques are often called *atomic commitment protocols*. In these techniques, there is necessary a network communication between all nodes to commit transaction. Such network interaction is known as voting. During voting each node votes if current transaction should be committed or rolled back. To commit given transaction there must be positive voting result. As an example of voting techniques there can be mentioned such methods as 2PC (*Two-Phase Commit*) — (Fig. 4), 3PC (*Three-Phase Commit*) or *quorum based* methods.

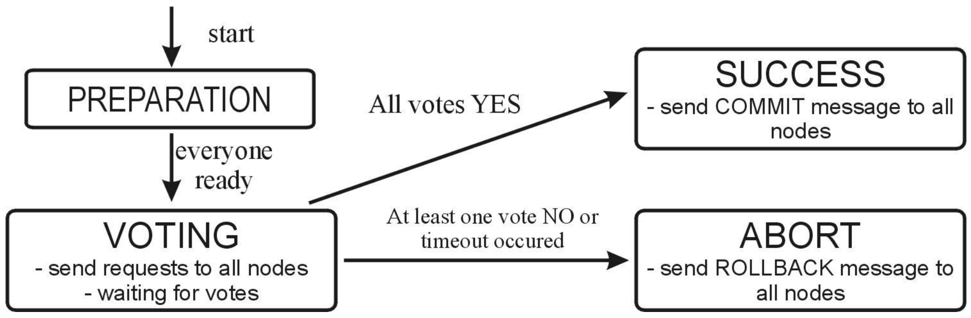
start
PREPARATION
everyone ready
VOTING
- send requests to all nodes
- waiting for votes
All votes YES
At least one vote NO or timeout occured
SUCCESS
- send COMMIT message to all nodes
ABORT
- send ROLLBACK message to all nodes

Fig. 4. Two-Phase Commit diagram

3.5.2. Non-voting techniques

In these techniques, each node alone decides about transaction commit. There is no need for network interaction such as voting between nodes. This method requires from nodes deterministic behavior. Because of this determinism each node can make decision by itself without communication or voting. Important issue in non-voting techniques are so-called *determinism points* in transactions. Determinism point is such a point in transaction execution, from which rest of transaction execution behave deterministically. Every transaction is executed only locally at current node. After reaching deterministic point is propagated to other nodes of distributed database where it is also executed.

3.6. Correctness criterion

This criterion estimates in what degree distributed database reflects the ideal model. The ideal case is when database meets 1-copy serializability and ACID (Atomicity, Consistency, Isolation and Durability). The first condition means that the result of execution of any sequence of operations (transactions) on distributed database is identical with the result of executing the same sequence of operations on single database. The second one touches among other things the isolation issue. In theory every transaction should be isolated from each other. Unfortunately, the ideal model is hard to implement in practice. Thus the most of available implementations offers only more restricted variants of this criterion, i.e. various isolations level defined by ANSI, but not only. Besides the ideal case (*Serializable*) there are used many other isolation levels, i.e. *Snapshot*, *Repetable Read* or *Cursor Stability*. Disadvantage of weaker isolation levels is possibility of occurrences of certain anomalies, such as *phantom reads* or *fuzzy reads*, but it is a compromise of efficiency and correctness. Detailed specification of correctness issue is presented in [5] and characteristics of basic isolation levels are shown in Table.

Isolation levels and their anomalies (legend: T — Yes, C — Sometimes)

Anomaly / Isolation level	Dirty Write	Dirty Read	Cursor Lost Update	Lost Update	Fuzzy Read	Phantom	Read Skew	Write Skew
Read Uncommitted	–	T	T	T	T	T	T	T
Read Committed	–	–	T	T	T	T	T	T
Cursor Stability	–	–	–	C	C	T	T	C
Repetable Read	–	–	–	–	–	T	–	–
Snapshot	–	–	–	–	–	C	–	T
Serializable	–	–	–	–	–	–	–	–

3.7. Platform diversity

Database platform diversity in distributed systems is a very important issue, especially in implementation phase. We distinguish two cases: homogeneous — which is very simple to implement and heterogeneous which implementation is much more complicated.

3.7.1. Homogeneous replication

This is a trivial case when all nodes represent an identical database platform. Because of this implementation of such system is simplified and has good efficiency. Most implementations of database replication systems work in homogeneous environment.

3.7.2. Heterogeneous replication

If nodes of distributed system operate on different database platforms, it is called heterogeneous replication. This case is much more complicated in its implementation. There are necessarily additional intermediate layers responsible for migration and transformation of data between servers. Because of large variety in interfaces, SQL syntax, data types, transaction log formats, and other issues there are many problems to solve. This significantly decreases distributed database efficiency in compare to homogeneous case.

4. Summary

In the given paper are presented elementary aspects of database replication and related issues. Obviously the area of the replication issue is very wide and it was not possible to discuss it in much detail. Furthermore, there are still created new techniques and algorithms of replication and these, which already exist, are still improved. Because of that, this paper cannot be considered as a comprehensive study on replication issue, but it only presents the main issues and problems related to database replication. Nevertheless these issues are quite universal and are connected with many much more advanced and complicated cases.

1. *Ciciani B., Dias D.M.* Analysis of Replication in Distributed Database Systems // IEEE Transactions on Knowledge and Data Engineering. — June, 1990. — Vol. 2, N 2.
2. *Gray J.N., Helland P., O'Neil P., and Shasha D.* The Dangers of Replication and a Solution // Proc. of the 1996 International Conf. on Management of Data. — Montreal (Canada). — June, 1996.
3. *Wiesmann M.* Group Communications and Database Replication: Techniques, Issues and Performance // Lausanne, EPFL. — 2002.
4. *Carrey M.J., Livny M.* Distributed Concurrency Control Performance: A Study of Algorithms, Distribution, and Replication. — Los Angeles (California). — 1988.
5. *Berenson H., Bernstein P.* A Critique of ANSI SQL Isolation Levels. — San Jose, 1995.
6. *Kemme B.* Database Replication for Clusters of Workstations. — Swiss Federal Institute of Technology (ETH). — Zurich, 2000.
7. *Lin Y.* Database Replication in Wide Area Networks. — 2005.
8. *Wiesmann M., Pedone F., Schiper A., Kemme B., Alonso G.* Database Replication Techniques: a Three Parameter Classification. — Swiss Federal Institute of Technology (ETH). — Zurich, 2000.

Received 05.03.2007