U.P.B. Sci. Bull., Series C, Vol. 74, Iss. 2, 2012 ISSN 1454-234x

DISTRIBUTED DATABASES IN THE DISTRIBUTED COMMITTEE-MACHINES ARHITECTURE

Valentin PUPEZESCU[1]

Prezenta lucrare analizează pentru prima dată influența pe care bazele de date distribuite o au asupra aplicațiilor de descoperire a cunoştințelor în baze de date. Se studiază implementarea efectivă a bazelor de date distribuite în sistemul de gestiune a bazelor de date SQL Server şi modul în care arhitecturile Data Mining de tip Committee-Machines sunt afectate de acestea. Studiul este benefic pentru domenii ca cel medical, astronomie, financiar, industria jocurilor, controlul traficului aerian etc.

This paper analyzes for the first time the influences of distributed databases on applications of knowledge discovery in databases. It studies the effective implementations of distributed databases in the SQL Server database management system and how they affect Data Mining architectures like Committee-Machines. The study is beneficial for areas such as medicine, astronomy, finance, gaming industry, air traffic control, and others.

Keywords: KDD, Knowledge Discovery in Databases, Data Mining, Replication, Knowledge Management, Distributed Committee-Machines

1. Introduction

Knowledge Discovery in Databases (KDD) is the overall process of finding and extracting useful patterns from data [1].

Data Mining (DM) represents a set of specific methods and algorithms aimed solely at extracting patterns from raw data [1].

In the last years the KDD process was approached from two perspectives: parallel and distributed computing. These directions led to the apparition of Parallel KDD and Distributed KDD [2]. In Parallel KDD, data sets are assigned to high performance multi-computer machines for analysis. The availability of this kind of machines is increasing and all algorithms that were used on single-processor units must be scaled in order to run on parallel-computers [2]. The Parallel KDD technology is suitable for scientific simulation, transaction data or telecom data [2]. Distributed KDD must provide solutions for local analysis of

[1]PhD student, Faculty of Electronics, Telecommunications and Information Technology, University POLITEHNICA of Bucharest, Romania, e-mail: vpupezescu@yahoo.com

data and global solutions for recombining local results from each computing unit without causing massive data transfer to a central server [2].

Although important studies were made in the field of distributed Data Mining (DDM), none of them was about the real implementations of distributed databases and how these implementations affect the overall performance of D-DM architectures.

In common studies that were made, data were kept on different locations (heterogeneous databases). That was the "distributed" meaning of distributed databases. As we will see, this view has little to do with real implementations of distributed databases.

In this paper I will focus on studying the influence that data replication has on a D-DM architecture. The analysis will treat the case of Distributed Committee-Machines architecture because for this type of structure, input data must be the same on all computers.

2. Distributed Databases

A distributed database (D-DB) is defined as a collection of multiple logically interrelated databases that are dispersed over a network of interconnected computers [3].

In a distributed database system, data is physically stored across several sites, and each site is typically managed by a database management system that is capable of running independently of the other sites [4]. The locations of data items and the degree of autonomy of individual sites have a significant impact on all aspects of the system, including query optimization and processing, concurrency control, and recovery [4]. So the DBMS carries out the replication of data and assures that data distribution is transparent to the user [3][4].

Microsoft SQL Server provides the following types of replication for use in distributed applications: merge replication, snapshot replication and transactional replication. There is also another type of replication called transactional replication with queued updating which is a variant of the transactional one. This type lets subscribers do updates to their databases.

In order to function in SQL Server, replication has five specialized agents:

- A distribution agent that moves the information from the distributor on the subscriber's databases [5];
- A log reader agent that monitors all the transaction logs from published databases that use it for replication. When the reader finds transactions that belong to a publication, it copies them to the distributor, from where the distribution agent can apply them to subscribers [5];
- A merge agent that combines the changes made in different locations [5];

- A snapshot agent that copies al the records from the publisher to subscribers [5];
- A copy agent from queue that reads messages from the SQL Server queue for every subscriber and applies the transactions at publisher's level [5].

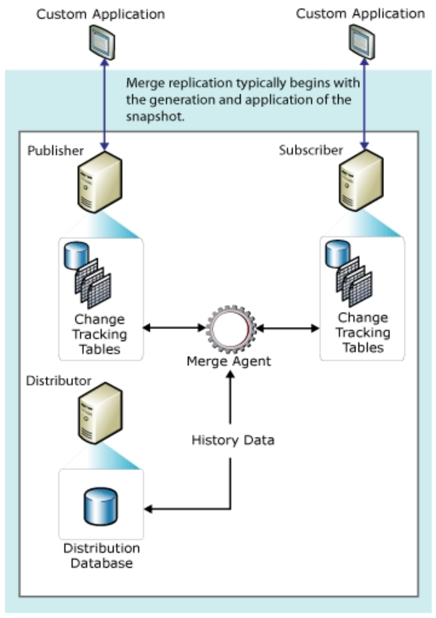

Fig. 1. Merge replication [6].

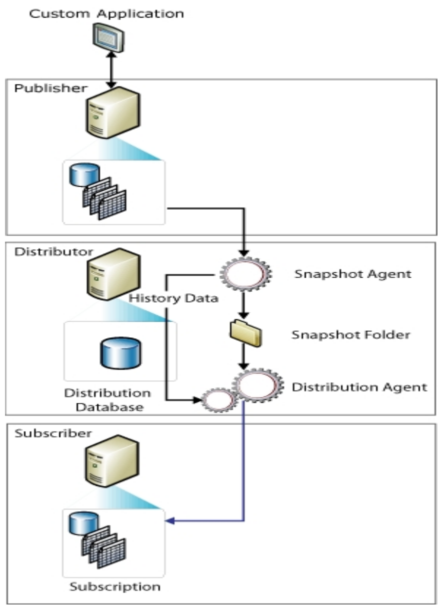

Fig. 2. Snapshot replication [6].

Merge replication (Fig. 1) offers the highest level of autonomy and accepts the highest levels of latencies.

In this type of replication we have the following latencies (these are also the steps of merge replication) [5]:

- $t_{snapshot}$ – the time period in which the snapshot agent(from the distributor) makes the initial copy of the database to each subscriber;
- t_{create} – the time in which a distribution folder is created on the distributor server. This is the folder were all the data will be combined. After this folder is created, the replication process starts;
- t_{merge1} – the time period in which the merge agent takes the changes from the publisher systems and applies them to subscribers;
- t_{merge2} – the time period in which the merge agent takes the changes from the subscribers and applies them to the publisher;

- t_{merge3} – the time in which the merge agent receives the update conflicts and takes the proper actions.

The actual value of replication process (measured in experiments) is:

$$TR_{Merge} = t_{snapshot} + t_{create} + t_{merge1} + t_{merge2} + t_{merge3} \tag{1}$$

Snapshot replication (Fig. 2) makes the copy of the entire database to all the subscribers [5]. This replication guarantees transactional consistency because all changes are made on the publisher system.

In this type of replication we have the following latencies (these are also the steps of snapshot replication) [5]:

- $t_{snapshot}$ – the time in which the snapshot agent reads the publisher article and creates the schema and table data in the distribution folder;
- $t_{distrib1}$ – the time in which the distribution agent reads this schema and reconstructs the table on the subscriber's system;
- $t_{distrib2}$ – the time in which the distribution agent transfers table data to the subscriber;
- t_{index} – the time for reconstructing indexes (if they are used) on the subscriber's system.

The actual value of replication process (measured in experiments) is:

$$TR_{Snapshot} = t_{snapshot} + t_{distrib1} + t_{distrib2} + t_{index} \tag{2}$$

In the transactional replication (Fig. 3) the transactions are sent from publisher to subscribers. Changes are made only at the publisher.

The time periods and steps for the transactional replication are shown below:

- $t_{snapshot}$ – the time period in which the snapshot agent creates the schema and table data in the distribution folder;
- $t_{distrib1}$ – the time in which the distribution agent reads this schema and creates the table on the subscriber's system;
- $t_{distrib2}$ – the time in which the distribution agent transfers the data on the subscriber. In this moment, the replication starts;
- t_{index} – the time for reconstructing indexes (if they are used) on the subscriber's system. In this moment, the replication of the transactions begins.
- t_{read} – the time in which the log reader agent checks transactional logs from the publisher. When it finds a transaction, it transfers it in the distribution database where it will be kept until the next synchronization starts;
- t_{sync} - when the synchronization it beeing made, the transaction it's read by the distribution agent and then it's executed on the subscriber's system.

The actual value of replication process (measured in experiments) is:

$$TR_{transact1} = t_{snapshot} + t_{distrib1} + t_{distrib2} + t_{index} + t_{read} + t_{sync} \tag{3}$$

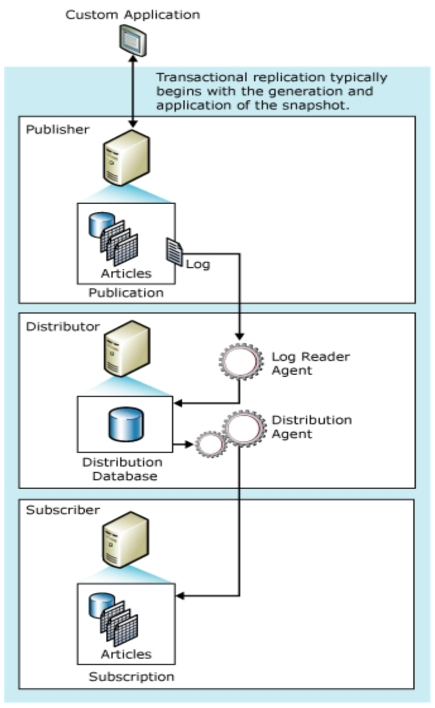

Fig. 3. Transactional replication [6].

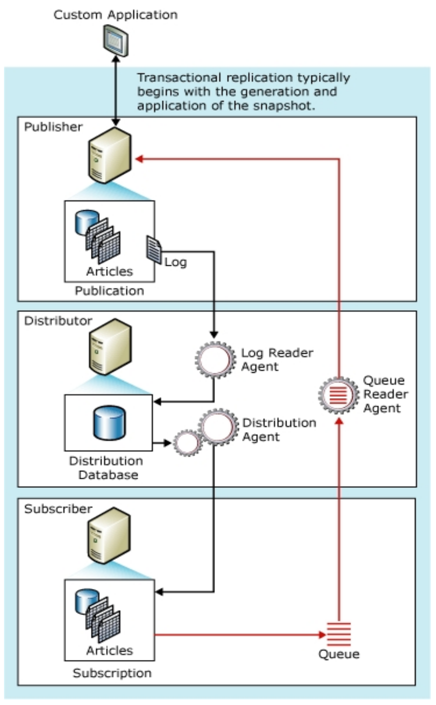

Fig. 4. Transactional replication with queued updating subscriptions [6].

Transactional replication with queued updating subscriptions (Fig. 4) is a variation of the classic transactional replication that allows subscribers to replicate changes to the publisher. The time periods and steps for the transactional replication with queued updating subscriptions are shown below [6]:

- t_{trigg} – the time in which updates made at the subscriber are taken by triggers on the subscribing tables. These updates are stored by trigger in the MSreplication_queue;
- t_{queue} – the time in which the queue reader agent reads the Msreplication_queue, and then applies transactions that are stored in it to the publication using replication stored procedures;
- $t_{resolve}$ – the time period in which conflict are resolved;
- t_{send} – the time period in which changes made at the publisher are propagated to all other subscribers according to the distribution agent schedule.

The actual value of replication process (measured in experiments) is:

$$TR_{transact2} = t_{trigg} + t_{queue} + t_{resolve} + t_{send} \tag{4}$$

3. Distributed Databases and D-CM architectures

A committee machine is a type of neural network system composed by many neural network structures that work in the same time and provide the best solution (choosen from all obtained results) to a given problem (Fig. 5).

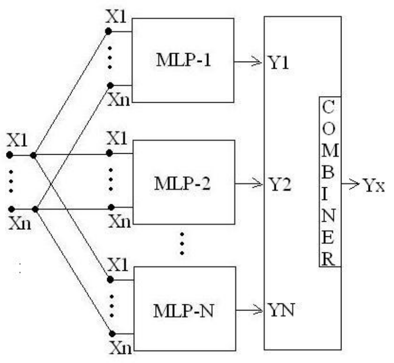

Fig. 5. Example of a multilayer committee-machine [2]

In the above figure it's presented an example of a D-CM architecture composed of multilayer perceptrons. All perceptrons work in a distributed manner and obtain the classification results. These results are then taken by the combiner. The combiner choose the best result from all the distributed perceptrons. Each neural structure from the above architecture works with a database that contains the input data so therefore it's very important to have the same imput data on all distributed systems. The best solution for doing this is using a type of replication presented in the previous chapter. If data is not replicated, each system can have different data for the perceptrons so the results would not be correct. Another unreliable solution would be to let the human factor to take care for copying data on all distributed system. This would be unprofitable. On the other hand, it would very good that all distributed structures to be aware of the results of the others because if a certain classifcation result would be obtained, all of them would stop working. The problem that appear in this situation are those of insert operations that are beeing made by all neural structures in the results tables.

Another problem that appears when working in a concurent environment is that of the transactional isolation level that was chosen. In all DBMS there are four transactional isolation levels: read uncommitted, read committed, repeatable reads and serializable. The first level is very permissive and the last one is the most restrictive (there, all the transactions will be treated in a serialized manner).

In the experiments, each distributed structure made 1000 insert operations.

The results took in the account the transactional isolation level.

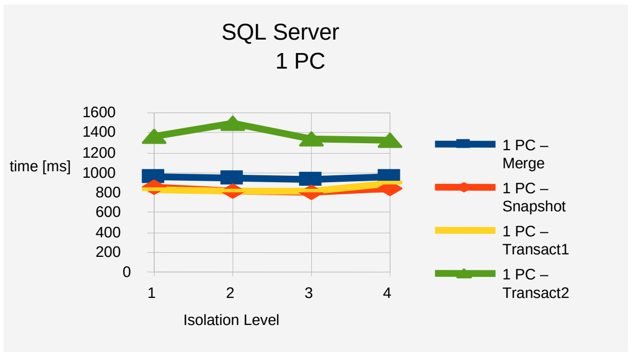

Fig. 6. 1000 inserts operations made by one PC linked to the publisher.

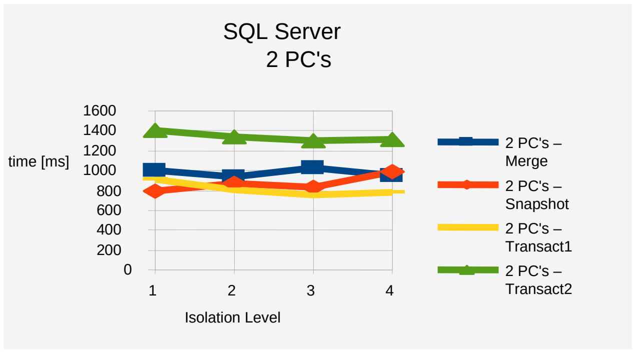

Fig. 7. 1000 inserts operations made by each of the two PCs linked to the publisher.

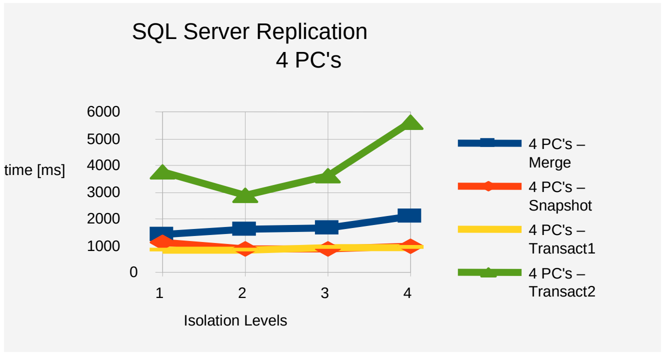

Fig. 8. 1000 inserts operations made by each of the four PCs linked to the publisher.

As we can see in the experimental results, the best results were obtained for the classic transactional replication. On all cases this replication didn't exceed 1 second. This is the most indicated replication type for distributed committee machines or for distributed concurrent neural networks. This good result are explained by the fact that when one machine from the system does an insert operation into the distributed database, the SQL Server runs the same insert instruction on each distributed client linked to the server without copying the entire database to each client.

The worst case was that of the transactional replication with updating queued. That happened because the listening process that it's taking place between all the systems in the network. The differences in terms of obtained times between these extreme cases are getting bigger as we grow the number of concurent systems in the D-CM structure.

The snapshot replication had also very good results in terms of insert operations but we must underline the fact that after the results are beeing transfered to the publisher, the network load is high because all the data are copied to each subscriber.

The transactional levels affected only the merge replication and the transactional replication with updating subscriptions. The more distributed systems are used in D-DM the lower isolation levels should be set in order to support more concurrent systems.

4. Conclusions

This paper is the first one that analyses the performances of insert operations of distributed committee-machines that are operating within a replication topology. The study have a very big importance for study fields like medical diagnosis, medical research, astronomy, finance and so forth.

With the results obtained we can say that in a distributed committee-machine architecture it is best to work with the classical transactional type of replication because of low execution times that were experimentally obtained.

R E F E R E N C E S

[1] *Usama Fayadd, Gregory Piatesky-Shapiro, and Padhraic Smyth*. From Data Mining To Knowledge Discovery in Databases, AAAI Press / The MIT Press, Massachusetts Institute Of Technology, 0-262-56097-6 FAYAP, 1996.

[2] *Pupezescu Valentin, Ionescu Felicia*. Advances in Knowledge Discovery in Databases, 6, 978-973-671-162-6, 2008.

[3] *Robert Dollinger*. Baze de date şi gestiunea tranzacțiilor (Databases and transaction management) Editura Albastră, 973-650-020-9, 2001.

[4] *O'Brien, J. & Marakas*, G.M. Management Information Systems (pp. 185-189). New York, NY: McGraw-Hill Irwin, 2008.

[5] *Richard Waymire, Rick Sawtell*. Microsoft SQL Server 2000. Editura Teora, 973-20-0468-1, 2002.

[6] *Help Microsoft* on-line, http://msdn.microsoft.com/en-us/library/bb500348.aspx.

[7] *Mukarram A.Tahir*, Java Implementation of Neural Networks, Booksurge Publishing Inc., 1997.