Concurrency Control in Distributed Database Systems[1]

IACOB, Nicoleta Magdalena
Spiru Haret University
Scientific Research Center in Mathematics and Computer Science
nicoleta.iacob_2007@yahoo.com

Abstract

In this paper are presented the methods to prevent and solve the problems related to distributed transactions that operate in parallel, access common data and eventually interfere one with another by using concurrency control algorithms implemented in Java programming language.

Keywords: *concurrency control, distributed database management systems, distributed transaction, data synchronization.*

ACM/AMS Classification: 68M14

1. ***Introduction***

In todays world of universal dependence on information systems, the rising need for secure, reliable and accessible information ([1], [2], [12]) to sustain business environment ([8], [9], [10, [11], [13]) led to the need of distributed databases that are able to support a large number of client/server applications.

Definition 1.1. Distributed Database Systems (DDBS) are defined as integrated database systems composed of autonomous local databases, geographically distributed and interconnected by a computer network. Every computer in the network has the autonomy to process local applications. Also, each computer participates in the execution stage of at least one global application that requires accessing data from multiple computers.

Definition 1.2. 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 ([7], [4]). The objective of transparency is to make a distributed system appear similar to a centralized system. This is usually called the *fundamental principle of DDBMS* [5].

Concurrency control is an important issue in database systems.

Concurrency control is the process of coordinating concurrent accesses to a database in a multi-user database management system (DBMS).

A **transaction** consists of a series of operations performed on a database. The important issue in transaction management is that if a database was in a consistent

[1]Citation: Iacob N.M., *Concurrency Control in Distributed Database Systems*, "An. Univ. Spiru Haret, Ser. Mat.-Inform.", 12(2), pp. 19-24, 2016.

state prior to the initiation of a transaction, then the database should return to a consistent state after the transaction is completed.

A **distributed transaction** is a transaction that runs in multiple processes, usually on several machines. Each process works for the transaction.

Definition 1.3. The **concurrency control in a distributed database** is the management of transactions that operate in parallel (the database is simultaneously accessed by many users), access common data and can eventually interfere one with another.

Whenever multiple users or programs access a database concurrently, the problem of concurrency control arises. The main technical difficulty in attaining this goal is to prevent database updates performed by one user from interfering with database retrievals and updates performed by another. When the transactions are updating data concurrently, it may lead to several problems with the consistency of the data. The problem is to synchronize concurrent interactions so that each reads consistent data from the database, writes consistent data, and is ultimately processed to completion.

Distributed concurrency control poses special challenges beyond centralized one, primarily due to communication and computer latency. The concurrency control problem is exacerbated in a DDBMS because:

- users may access data stored in many different computers in a distributed system;
- a concurrency control mechanism at one computer cannot instantaneously know about interactions at other computers.

In case of a simultaneous update to an object performed by many users, concurrency controlling techniques are used. Concurrency controlling techniques ensure that multiple transactions are executed simultaneously while maintaining the ACID (Atomicity, Consistency, Isolation, Durability) properties of the transactions and serializability in the schedules.

There exist a number of methods that provide concurrency control:

- *locking (two phase locking – 2PL)*: Centralized 2PL, Primary copy 2PL, Distributed 2PL, Voting 2PL;
- *timestamp ordering*;
- *optimistic*;
- *hybrid.*

2. ***Concurrency control algorithm***

Every client has a table b of elements from $\{ 0, 1 \}$, where $b[i] = 1$ if and only if the client has in his database object i. The clients are connected to the same server that does not have an owned database. The activity of a client i consists of requesting a specific object. For this request, it transmits to the server the object number. The server queries the clients and determines a client j that has this object. Client j returns the object to the server and the server returns the object to client i.

A client receives data in two contexts:

- as a result of his primary request of receiving an object;
- as a server request received to satisfy the primary request of another client.

To avoid overlapping this actions (concurrency between primary actions and concurrency between a primary action and an inter-clients action) it is necessary to realize a mutual exclusion. The solution consists of connecting every client to server using two separate threads ([3], [6]):

- one thread allocated exclusively to primary request and, by mutual exclusion, does not permit another client to connect to server for a similar request;
- the second thread to manage the object flow between clients.

Similarly, on the server are started two execution threads for solving the two actions described above on ports 1234, respectively 4321.

The class *Client* includes the client activity:

- on the Threadst execution thread, corresponding to graphical interface, the client is connecting to server on port 1234 for primary requests;
- on the second execution thread, the connection to server is established on port 4321 for the second category of requests.

Sockets are used for connections.
The server activities are described like this:

- it is mentioned Threadst the class *Elem* which encapsulates for every client the socket where is connected and the related flows, the name of the client and table b that contains the client objects index.

The main class is *Server* class, which starts an execution thread (type S1) for the communication through port 1234 and an execution thread (type S2) for the communication on port 4321.

The execution thread of type class *S1* creates in a collection *al* of type *ArrayList* an object of type *Elem* for every client and starts an execution thread of type *ServerThread1* for every client. The main action, which corresponds to method *transaction*, performs mutual exclusion through synchronization of object *al*. Additionally, the case in which a client disconnects from server is taken into account.

```
void transaction() {
synchronized(al) { while(Server.i != -1); }
}
void removeConnection(Elem elem) {
Elem e;
synchronized(al) {
System.out.println("Delete connection with " + elem.name);
for(Iterator i = al.iterator(); i.hasNext(); ) {
e = (Elem) i.next();
if(e.cs == elem.cs) {
try { e.cs.close(); } catch(IOException w) { }
break; }
}}}
```

In class *ServerThread1* are stipulated the following actions: read of a primary request, the waiting time to solve it and the return of a message (the equivalent of sending the requested object).

```
class ServerThread1 extends Thread {
S1 server;
Elem elem;
ServerThread1(S1 server, Elem elem) {
this.server = server;
this.elem = elem;
start();
}

public void run() {
try {
while(true) {
int i = Integer.parseInt(elem.dis.readUTF());
Server.i = i;
System.out.println(elem.name + " request: " + i);
server.transaction();
System.out.println("Send to " + elem.name + " object " + i);
elem.dos.writeInt(i);
}
} catch(EOFException e) { }
catch(IOException e) { }
finally { server.removeConnection(elem); }
}
}
```

The execution thread of type class *S2* creates in a collection al of type *ArrayList* an object of type *Elem* for every client and starts an execution thread of type *ServerThread2* for every client. The main action, which correspond to method *search*, returns the information about the client which contains the object requested in primary request. Additionally, the case in which a client disconnects from server is taken into account.

```
Elem search(int j) {
Elem e;
ObjectOutputStream dos;
for(Iterator i = al.iterator(); i.hasNext(); ) {
e = (Elem) i.next();
if(e.b[j]==1) return e;
}
return null;
}

void removeConnection(Elem elem) {
Elem e;
synchronized(al) {
.....
Similar with action from S1, but on port 4321
..... }}
```

```
class ServerThread2 extends Thread {
S2 server;
Elem elem;
ServerThread2(S2 server, Elem elem) {
this.server = server;
this.elem = elem;
start();
}

public void run() {
int i;
try {
while(true) {
while(Server.i==-1) ;
i = Server.i;
Server.i = -1;
System.out.println("Looking for a client that has the object " + i);
elem = server.search(i);
System.out.println("I found the object " + i + " to " + elem.name);
elem.dos.writeInt(i);
System.out.println("I sent to " + elem.name + " request ");
System.out.println("I received from " + elem.name + " object " + elem.dis.readInt());
try {
Thread.sleep(50); } catch(Exception e) { }
Server.i = -1;
}
} catch(EOFException e) { }
catch(IOException e) { }
finally { server.removeConnection(elem); }
}}
```

In class *ServerThread2* are stipulated the following actions: the identification of a client that owns the requested object, the request to this client, the answer received and the return of a message (the equivalent of sending the object to the client that sent the primary request). Additionally, the client that initiated the request is unlocked on port 1234.

3. ***Conclusion***

In this paper is described the concept of concurrency control in distributed databases, showing the methods to avoid and solve database updates performed by one user from interfering with database retrievals and updates performed by other users which are simultaneously accessing the database. The unexpected events generated by concurrent users that access the databases are solved using concurrency control algorithms implemented in Java programming language.

References

1. C.L. Defta, *Information Security in E-learning Platforms*, ”Procedia Social and Behavioral Sciences”, Elsevier, Vol. 15, pp. 268-269, 2011.
2. C.L. Defta (Ciobanu), *Security Issues in e-Learning Platforms*, ”World Journal on Educational Technology”, Vol. 3, issue 3, 2011, pp. 153-167.
3. H.I. Georgescu, *Introduction to Java/Introducere în universul Java*, Technical Publishing House/Editura tehnică Bucharest, Romania, 2002.
4. R. Elmasri, S. Navathe, *Fundamentals of database systems* (4th ed.), Boston: Addison-Wesley, 2004.
5. N.M. Iacob (Ciobanu), *Queries in Distributed Databases*, ”Global Journal on Technology Science”, Vol. 1, 2012, pp. 142-147.
6. N.M. Iacob, *Concurency Problems in Distributed Databases*, ”Proceedings of the 9th International Conference on Virtual Learning 2014. Models & Methodologies, Technologies, Software Solutions”, University of Bucharest, Faculty of Psychology and Educational Sciences, Siveco Romania, October 24-25, 2014, pp. 413-419.
7. M.T. Ozsu, P. Valduriez, *Principles of Distributed Database Systems*. (3th ed.), New York: Springer, 2011.
8. D.A. Popescu, N. Bold, *Web application presentation of timetable for a university website*, ”The 8th International Conference on Virtual Learning”, October 25 – 26, 2013, Models and Methodologies, Technologies, Software Solutions, Bucharest, Romania, Ed. Univ. Bucuresti, pp. 253-256.
9. D.A. Popescu, G. Boroghina, (2015a), *Students’ Computer-Based Distribution in Highschool Based on Options as an Extension of SEI Distribution Program*, ”The 10th International Conference on Virtual Learning”, October 30-31, 2015, Models and Methodologies, Technologies, Software Solutions, Timisoara, Romania, pp. 329-334.
10. D.A. Popescu, G. Boroghina, (2015b), *Web-Based Programming Model*, ”6th International Conference on Modeling, Simulation, and Applied Optimization (ICMSAO’15)”, May 27-29, 2015, Istanbul, Turkey, IEEE Xplorer, pp. 1-4.
11. D.A. Popescu, D. Radulescu, *Approximately Similarity Measurement of Web Sites,* ”22th International Conference on Neural Information Procession”, Nov. 09-12, 2015, Istanbul, Turkey, Springer Proceedings, LNCS, Part IV, pp. 624-630.
12. D.M. Radulescu, V. Radulescu, *Sustainable Development in Terms of Interpreting the Human Right to a Healthy Environment*, ”The Romanian Economic Journal”, Volume XV no. 46 bis, December 2012, pp. 111-120.
13. M.A. Titu, C. Pirnau, M. Pirnau, *Creativity, Education and Quality for Sustainable Development, the real Support for the Innovative Clusters European Network,* ”8th Research/Expert Conference with International Participations, QUALITY 2013”, 2013/6, pp.19-24.