CS542 - Distributed Database Systems

Project Report

Implement Centralized Two Phase Locking (2PL)

Andres Bejarano
Department of Computer Science
abejara@purdue.edu

April 21, 2016

Statement of the Problem

The level of concurrency is one of the most important parameters on a distributed database systems. A considerable number of control mechanisms have been proposed in the topic, mostly of them attempt to find a suitable trade-off between maintaining the consistency while maintaining a high level of concurrency. Traditionally, such mechanisms consider one of the following options for handling concurrency: locks, time stamps and versions. The lock approach is simple to implement; however, their optimal management is not a trivial task.

A vastly studied lock-based approach is Two-Phase Locking (2PL). This method states that no transaction should request a lock after it releases one of its locks. Alternatively, a transaction should not release a lock until it is certain that it will not request another lock. 2PL executes transactions in two phases: *growing phase* (locks and access to data items are obtained) and *shrinking phase* (locks are released and data items become available again). A middle step named *lock points* is the moment when a transaction has achieved all the required locks but none of them have been released. An important proven fact about 2PL is that any generated history under the 2PL rules is serializable.

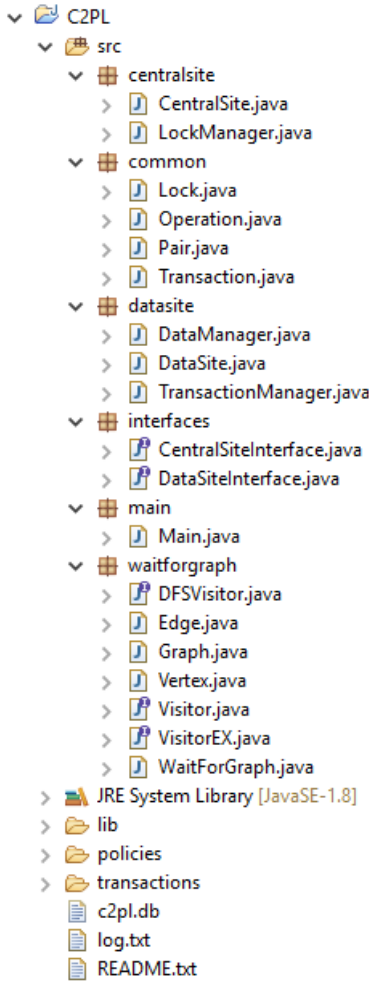
Methods or Procedures

Project Structure and technical information

This Centralized Two-Phase Locking implementation was coded in Java. Communications between the central site and the other sites (called data sites in this project) was implemented using Java RMI. The structure of the project (figure 1) consists on six main modules:

1. **interfaces:** Consist on two interfaces, CentralSiteInterface and DataSiteInterface. Both are required for the Java RMI correct functionality. Both interfaces define the methods that will me called remotely to the central site and the data sites respectively.

Figure 1: Structure of the project



2. **common:** Holds the common classes for all the sites in the system.

3. **waitforgraph:** Contains the files required for building waits for graphs. It is based on the WaitForGraph module included in the JBoss API.

4. **centralsite:** Defines the logic required for the central site following the C2PL specifications. It includes the lock manager functionality for managing the locks over the items in the database.

5. **datasite:** Defines the logic required for the data sites following the C2PL specification. For each data site there is a data manager and a transaction manager.

6. **main:** For testing purposes. It is a simple file that reads the contents of the database. It is used for checking the correctness of the implementation.

Each data site implements its own data manager. For this project it is a handler to a SQLite3 database managed using the sqlite-jdbc driver (found in https://bitbucket.org/xerial/sqlite-jdbc). The write operations include creating the database, inserting the item (it is the first time the data

item is operated) and updating the value of the data item. The write operation is the classical select operation used in SQL. Mathematical operations are allowed in the transactions. They require two operands per calculation, which include addition, subtraction, multiplication and division.

The overall implementation is based on the C2PL-TM (Figure 2), C2PL-LM (Figure 3) and DP (Figure 4) algorithms from section 11.3.1 from "Principles of Distributed Database Systems" book. The messages mechanism was replaced for remote method invocations, making the logic more straightforward to follow.

Assumptions and initial considerations

Based on the project plan submitted at the beginning of the semester, the following are assumptions and considerations of the project:

1. It is assumed the distributed system is fully reliable; therefore, failures are not expected.
2. Virtual sites will be generated and inserted to the network. Transactions are defined using text files.
3. A lock table and a wait-for graph (dependency graph) is maintained in the central site. All transactions arriving from the different sites are sent to the centralized site. Queues for requesting and releasing locks are maintained at this central site.
4. Request processing will take place at the data site where transaction is submitted. Internal 2PL information is displayed on the site's terminal interface.
5. Locks will be released after waiting on the database at the centralized site.
6. Updates for other sites will be sent and they may arrive in different order. It is assumed a fully replicated database.
7. All the updates are ensured at all sites and they are posted in the same order.

Running the Sites and Input Parameters

Running the central site requires the initialization of the RMI register service, indicating the location of the interfaces with the remote methods definitions. In the terminal console it is required to access to the main C2PL folder and type the following command:

```
start rmiregistry -J-Djava.rmi.server.codebase=file:bin\
```

Once the rmiregistry service is running then the central site can be executed. It is required to specify the following information:

- The classpath of the project, including the location of the sqlite-jdbc driver.
- The security policy for the central site (required for Java RMI).

Figure 2: C2PL-Transaction Manager Algorithm

Algorithm 11.1: Centralized 2PL Transaction Manager (C2PL-TM) Algorithm

```
Input: msg : a message
begin
    repeat
        wait for a msg ;
        switch msg do
            case transaction operation
                let op be the operation ;
                if op.Type = BT then DP(op)          {call DP with operation}
                else C2PL-LM(op)                     {call LM with operation}
            case Lock Manager response               {lock request granted or locks
            released}
                if lock request granted then
                    find site that stores the requested data item (say H_i) ;
                    DP_Si(op)                        {call DP at site S_i with operation}
                else                                 {must be lock release message}
                    inform user about the termination of transaction
            case Data Processor response             {operation completed message}
                switch transaction operation do
                    let op be the operation ;
                    case R
                        return op.val (data item value) to the application
                    case W
                        inform application of completion of the write
                    case C
                        if commit msg has been received from all participants
                        then
                            inform application of successful completion of
                            transaction ;
                            C2PL-LM(op)              {need to release locks}
                        else        {wait until commit messages come from all}
                            record the arrival of the commit message
                    case A
                        inform application of completion of the abort ;
                        C2PL-LM(op)                  {need to release locks}
    until forever ;
end
```

Figure 3: C2PL-Lock Manager Algorithm

Algorithm 11.2: Centralized 2PL Lock Manager (C2PL-LM) Algorithm

```
Input: op : Op
begin
    switch op.Type do
        case R or W                          {lock request; see if it can be granted}
            find the lock unit lu such that op.arg ⊆ lu ;
            if lu is unlocked or lock mode of lu is compatible with op.Type
            then
                set lock on lu in appropriate mode on behalf of transaction
                op.tid ;
                send "Lock granted" to coordinating TM of transaction
            else
                put op on a queue for lu
        case C or A                          {locks need to be released}
            foreach lock unit lu held by transaction do
                release lock on lu held by transaction ;
                if there are operations waiting in queue for lu then
                    find the first operation O on queue ;
                    set a lock on lu on behalf of O ;
                    send "Lock granted" to coordinating TM of transaction
                    O.tid
            send "Locks released" to coordinating TM of transaction
end
```

Figure 4: Data Processing Algorithm

Algorithm 11.3: Data Processor (DP) Algorithm

```
Input: op : Op
begin
    switch op.Type do                        {check the type of operation}
        case BT                              {details to be discussed in Chapter 12}
            do some bookkeeping
        case R
            op.res ← READ(op.arg) ;          {database READ operation}
            op.res ← "Read done"
        case W                               {database WRITE of val into data item arg}
            WRITE(op.arg, op.val) ;
            op.res ← "Write done"
        case C
            COMMIT ;                         {execute COMMIT }
            op.res ← "Commit done"
        case A
            ABORT ;                          {execute ABORT }
            op.res ← "Abort done"
    return op
end
```

- The hostname of the server. It could be a URL to a web server.
- The class containing the main function of the central site.
- The port where the remote method invocations will be sent.
- The time delay for deadlocks to be checked.

For instance, running the central site in localhost with port 45 and checking deadlocks every 3000 milliseconds then the command is the following:

```
java -classpath bin;lib\sqlite-jdbc-3.8.11.2.jar
-Djava.security.policy=policies\server.policy
-Djava.rmi.server.hostname=localhost centralsite.CentralSite 45 3000
```

Both the sqlite-jdbc driver and the policy file are included in the project folders. If it is not interested in defining the hostname then the command can be reduced to:

```
java -classpath bin;lib\sqlite-jdbc-3.8.11.2.jar
-Djava.security.policy=policies\server.policy centralsite.CentralSite
45 3000
```

For the data sites, the same approach is followed. It is required to run the central site class specifying the security policy for the client, the central site IP address, the listening port and the transactions file. Open a new terminal console and access the main C2PL folder, then type the following command:

```
java -classpath bin;lib\sqlite-jdbc-3.8.11.2.jar
-Djava.security.policy=policies\client.policy
-Djava.rmi.server.codebase=file:bin\datasite.DataSite localhost 45
transactions\transactions_long_1.txt
```

If the central site is running in a different machine then, instead of localhost insert the IP address or the URL of the central site.

Limitations of the System

In order to generate easy deadlocks it is required to have long transactions with multiple operations. For doing this it was designed different transactions files with repetitive operations over the same data items for increasing the chances of a deadlock. Four different transactions files with different lengths are included in the project (see the transactions folder). Once the data site finishes with the operations in the file it goes to a blocked state. Running again the site requires to finish the process and restart it using the same initial command.

A second limitation occurs while releasing the locks. The strict 2PL version was implemented for increasing the chances of a deadlock. Although a non-strict version could be developed, it requires non-trivial modifications of the code in the transaction manager module..

The last limitation concerns to aborted transactions. Once a transaction is aborted it is not executed again. This was done in order to follow the specification of the classic 2PL. Although termination methods are documented in literature, none of them were implemented.

Data Collected

A number of experiments were executed in order to test its correctness. Such experiments include solving deadlocks and data replication, running a considerable number of transaction requests for expected long wait times.

The first experiment consists on checking the correctness of the system on the presence of deadlocks. A central site and several data sites were run at the same time. Each data site run the same transactions file with the purpose of increasing the chances of deadlocks. It was found that the system worked as intended. Due to the length of the outputs, only a fragment of the output for two data sites are shown in this document. To see the full output of an experiment using long transactions see files in the logs folder.

The following output snippets shows a deadlock between two transactions from two different data sites. Data site 1 registers the deadlock as follows:

```
[2016-04-13 22:38:43.882] datasite.DataSite:  Initiate a new cycle
[2016-04-13 22:38:43.883] datasite.DataSite:  Transaction obtained
[2016-04-13 22:38:43.885] Starting transaction [T101]
[2016-04-13 22:38:47.242] Transaction 101 completed
[2016-04-13 22:38:47.244] datasite.DataSite:  Initiate a new cycle
[2016-04-13 22:38:47.246] datasite.DataSite:  Transaction obtained
[2016-04-13 22:38:47.247] Starting transaction [T102]
[2016-04-13 22:38:47.485] Site 1 blocked.  Waiting for events...
[2016-04-13 22:38:48.488] Site 1 blocked.  Waiting for events...
[2016-04-13 22:38:49.489] Site 1 blocked.  Waiting for events...
[2016-04-13 22:38:50.491] Site 1 blocked.  Waiting for events...
[2016-04-13 22:38:50.855] datasite.DataSite:  Preparing to unblock the
site
[2016-04-13 22:38:50.857] datasite.DataSite:  Site is now unblocked
[2016-04-13 22:38:54.540] Transaction 102 completed
[2016-04-13 22:38:54.542] datasite.DataSite:  Initiate a new cycle
[2016-04-13 22:38:54.543] datasite.DataSite:  Transaction obtained
[2016-04-13 22:38:54.545] Starting transaction [T103]
[2016-04-13 22:38:54.784] Site 1 blocked.  Waiting for events...
[2016-04-13 22:38:55.786] Site 1 blocked.  Waiting for events...
[2016-04-13 22:38:56.788] Site 1 blocked.  Waiting for events...
```

```
[2016-04-13 22:38:56.856] datasite.DataSite:  Preparing to unblock the
site
[2016-04-13 22:38:56.859] datasite.DataSite:  Site is now unblocked
[2016-04-13 22:39:00.835] Transaction 103 completed
```

The data site 2 (who started later than the data site 1) generated two deadlocks with its first two transactions. Data site 2 registers the deadlock and aborted transactions as follows:

```
[2016-04-13 22:38:45.96] datasite.DataSite:  Initiate a new cycle
[2016-04-13 22:38:45.98] datasite.DataSite:  Transaction obtained
[2016-04-13 22:38:45.99] Starting transaction [T201]
[2016-04-13 22:38:45.129] Site 2 blocked.  Waiting for events...
[2016-04-13 22:38:46.131] Site 2 blocked.  Waiting for events...
[2016-04-13 22:38:47.133] Site 2 blocked.  Waiting for events...
[2016-04-13 22:38:47.222] datasite.DataSite Writing the info for site 2
[2016-04-13 22:38:47.226] datasite.DataSite:  Preparing to unblock the
site
[2016-04-13 22:38:47.228] datasite.DataSite:  Site is now unblocked
[2016-04-13 22:38:48.354] Site 2 blocked.  Waiting for events...
[2016-04-13 22:38:49.357] Site 2 blocked.  Waiting for events...
[2016-04-13 22:38:50.358] Site 2 blocked.  Waiting for events...
[2016-04-13 22:38:50.827] datasite.DataSite:  Site is now aborted
[2016-04-13 22:38:50.829] datasite.DataSite:  Preparing to unblock the
site
[2016-04-13 22:38:50.833] datasite.DataSite:  Site is now unblocked
[2016-04-13 22:38:51.359] Transaction 201 is aborted
[2016-04-13 22:38:51.361] datasite.DataSite:  Initiate a new cycle
[2016-04-13 22:38:51.365] datasite.DataSite:  Transaction obtained
[2016-04-13 22:38:51.367] Starting transaction [T202]
[2016-04-13 22:38:51.386] Site 2 blocked.  Waiting for events...
[2016-04-13 22:38:52.388] Site 2 blocked.  Waiting for events...
[2016-04-13 22:38:53.389] Site 2 blocked.  Waiting for events...
[2016-04-13 22:38:54.391] Site 2 blocked.  Waiting for events...
[2016-04-13 22:38:54.517] datasite.DataSite Writing the info for site 2
[2016-04-13 22:38:54.521] datasite.DataSite:  Preparing to unblock the
site
[2016-04-13 22:38:54.525] datasite.DataSite:  Site is now unblocked
[2016-04-13 22:38:55.609] Site 2 blocked.  Waiting for events...
[2016-04-13 22:38:56.611] Site 2 blocked.  Waiting for events...
```

```
[2016-04-13 22:38:56.827] datasite.DataSite:  Site is now aborted
[2016-04-13 22:38:56.830] datasite.DataSite:  Preparing to unblock the
site
[2016-04-13 22:38:56.834] datasite.DataSite:  Site is now unblocked
[2016-04-13 22:38:57.614] Transaction 202 is aborted
[2016-04-13 22:38:57.616] datasite.DataSite:  Initiate a new cycle
[2016-04-13 22:38:57.619] datasite.DataSite:  Transaction obtained
[2016-04-13 22:38:57.621] Starting transaction [T203]
[2016-04-13 22:38:57.643] Site 2 blocked.  Waiting for events...
[2016-04-13 22:38:58.645] Site 2 blocked.  Waiting for events...
[2016-04-13 22:38:59.647] Site 2 blocked.  Waiting for events...
[2016-04-13 22:39:00.648] Site 2 blocked.  Waiting for events...
[2016-04-13 22:39:00.814] datasite.DataSite Writing the info for site 2
[2016-04-13 22:39:00.819] datasite.DataSite:  Preparing to unblock the
site
[2016-04-13 22:39:00.824] datasite.DataSite:  Site is now unblocked
[2016-04-13 22:39:04.929] Transaction 203 completed
```

It can be seen that transactions 201 and 202 in data site 2 are aborted by the central site over a conflict with the same data item. Sites remain in a blocked state when the lock manager cannot process then. Checking deadlocks is performed every three seconds in the central site. Once a deadlock is detected, the most recent transaction is aborted, allowing the previous transaction to finish its execution. For the registered output of the central site please take a look at the CentralSiteLog.txt file in the logs folder.

The second experiment focused on testing the robustness of the system in the presence of a massive number of transactions. A central site and two data sites were configured, each data site running a transactions file (transactions\transactions_many_1.txt) consisting of 7200 transactions. It took around 3 hours and 45 minutes to finish them all. No crashing or exception occurred during this test.

Experiences

A serious technical problem encountered at the beginning of the development was running correctly the application using the RMI specification in the Oracle web page. Although the code's syntax was OK, the project was hard to run since the instructions given in the official web page are out of date for the current Java RMI implementation (the project was coded using Java 8).

Finding a solution for such problem was a time consuming task. It was found the solution in the Stack Overflow web page. The main problem was the definition of the codebase in the central site. Traditional RMI initialization specifies such codebase to be defined as a parameter of the main class using `-J-Djava.rmi.server.codebase=file:bin\`. However, it was found a best practice

is to initialize the rmiregistry service with such parameter. After such modification the central site executed without problems.

An interesting aspect of the project was using Java RMI instead of Sockets and ServerSockets. The decision was more as a personal challenge for learning more about remote method invocations. Since the original 2PL algorithms describe a message interchange between sites, translating the logic to method invocations was an interesting task. I found this mechanism quite powerful since the developer doesn't have to worry about communication issues and data transfers, everything works with the logic of a stand-alone application.

Observations

The current implementation requires additional modifications for large deployments:

- Data sites require an additional input mechanism for new transactions to run once the previous ones were fully executed.
- Aborted transactions should be stored for later executions.
- Non-strict 2PL implementation is required for flexible executions. This modification can be implemented in the lock manager module.
- Using URLs as host names was not tested. This aspect is important since most web servers rely on web services with sharing resources through this mechanism.
- A more professional user interface (for site management) is important for handling different configurations in the system. For now the system requires input parameters in the executing command.

For sure this project could be related to multiple independent data base managers for specific purposes. Projects such as SQLite could be enhanced by this simple C2PL implementation (due to its low memory requirements and quick integration with the database process).

No doubt this serializability mechanism is good enough for small transactions. However, large transactions are severely affected by deadlocks, and therefore, aborts. I consider that ideas about lock granularity are required to be adopted in more general formulations of the protocol. Also, having an indicator for read-only and write-only transactions might help considerably in the right serialization process. We could save time an processing resources if both the lock manager and the transaction manager are capable of differentiating and handling such transactions once one of them arrives for execution.

References

M. T. zsu, Principles of Distributed Database Systems, 3rd edn. (Springer-Verlag, Berlin, Heidelberg, 2011).

Alsberg, P. A. and Day, J. D. (1976). A principle for resilient sharing of distributed resources. In Proc. 2nd Int. Conf. on Software Engineering, pages 562570.