Design Document - Replicated Concurrency Control and Recovery

Sanchit Mehta and Pranav Chaphekar

24 October 2017

Abstract

In this project, we are implementing the distributed database, complete with multiversion concurrency control, deadlock detection, replication and failure recovery. We aim to get the deep insight into the distributed systems and databases with this project.

1 Introduction

We are implementing the available copies approach to replication using strict two phase locking protocol at each site nad validation at commit time. A transaction may read a variable and later write that same variable as well as others. Along with this, we are also doing the deadlock detection. The deadlock detection is done by cycle detection and we must always abort the youngest transaction. So, our system keeps the track of the oldest transaction time of any transaction holding a lock. Moreover, read-only transactions use multi-version read consistency.

2 Theory

We know that distributed database is a collection of multiple interconnected databases, which are spread physically across various locations that communicate via a computer network. In this project, we are replicating these databases on 20 locations i.e. sites. The data is stored at each site, and in the currrent scenario we have the data in the form of the 20 variables at each site. All the transactions and sites are managed by transaction manager. It is important that the integrity of the database is maintained, by which we mean that all the sites that are up, will have the same data, and at a very high level, it will give an impression to the user that only one site exists.

Another important aspect in the project is the database recovery.Replication of data automatically helps in data recovery if database in any site is damaged. Users can access data from other sites while the damaged site is being reconstructed. Thus, database failure may become almost inconspicuous to users.

In order to maintain the data consistency and integrity, the transaction must acquire the locks before it can execute. There are 2 types of locks - Read (Shared) lock and Write

(Exclusive) lock. Only one transaction cah have a write lock on some variable, but multiple transactions can have a read lock on the same variable.
The locks are acquired by the transaction in the manner of strict 2pl, which has the growing phase and the shrinking phase. Hence, before the transaction ends, it writes and reads all the values of the variables. If the transaction does not have any of the lock by then, it gets aborted. If there is a cycle in the graph, then the youngest transaction is killed. Once, the transaction is aborted, it is never restarted.

3 Modules

3.1 Transaction

Contains all the information about the transactions. The information includes the transaction id, the start time of the transaction and the various operations that have been included in the transaction

3.2 Read Only Transaction

This is a subclass of the transaction, and it includes the state(values) of the variables at the start of the transaction, which will be used henceforth in the till the transaction commits / aborts. This is because of the multi-version read concurrency protocol.

3.3 Input Parser

The input parser reads the input line by line, understands it and calls the transaction manager. Following are the various types of input:

1. *begin(T1)*- This means that Transaction T1 begins and it is a normal transaction

2. *beginRO(T2)* - This means that Transaction T2 begins and it is a read only transaction

3. *R(T1, x4)* - This is a read operation, it means that transaction T1 reads the value of variable x4 from any site, provided it gets the lock. However, if T1 is a Read only operation then it might not need the lock. This statement if successful just prints the value

4. *W(T1, x6,v)* - This is a write operation, which states that transaction T1 will update the value of variable x6 to v.

5. *dump()* - Prints values of all the variables at all the sites in the sorted order (tabular format)

6. *end()* - This ends the transaction, i.e. either commits or aborts the transaction.

7. *fail(site Id)* - Fails the site given by site Id. All variables that are present on this site have to give up all the read and write locks

8. *recover(site Id)* - Recovers the site defined by the site Id. When the site is recovered, the variables on the site gets initialized by all the values which are present on the other sites.

3.4 Dump Output

This class displays the output in the tabular format

3.5 Transaction Manager

Transaction Manager is the one which controls the processing of all the transactions at all the sites. It never fails. The Transaction Manager has all the informations of all the transactions that have begun, ended, aborted and that are running currently.

Transaction Manager maintains a waitlist queue which contains the ids of the transactions that are currently waiting for the locks. Along with this, it maintains the queue of the pending operations, which will be execueted as soon as they obtain the requisite locks.

3.6 Deadlock Handler

One of the most important aspect of the project is to detect the deadlock and if deadlock exists then abort the youngest transaction in the cycle. The deadlock handler class consists of wait for graph and wait by graph. In case of wait for graph, if we have the directed edge from Transaction T1 to T2, it means that transaction T1 is waiting for T2 to complete. Other way round, an edge from T2 to T1 in wait by graph indicates that T2 is causing T1 to wait.

Thus, the deadlock detection will find the loop in the wait for graph and break the loop by killing the youngest transaction.

3.7 Site

This contains the information of all the variables on the site with their values. This is helpful for the fail and recovery scenario.

3.8 Variable

This class contains all the information about the variable. The information includes the data about all the read locks and write locks that are acquired by the variable

4 Design Diagram

The following diagrams illustrates the design of the system

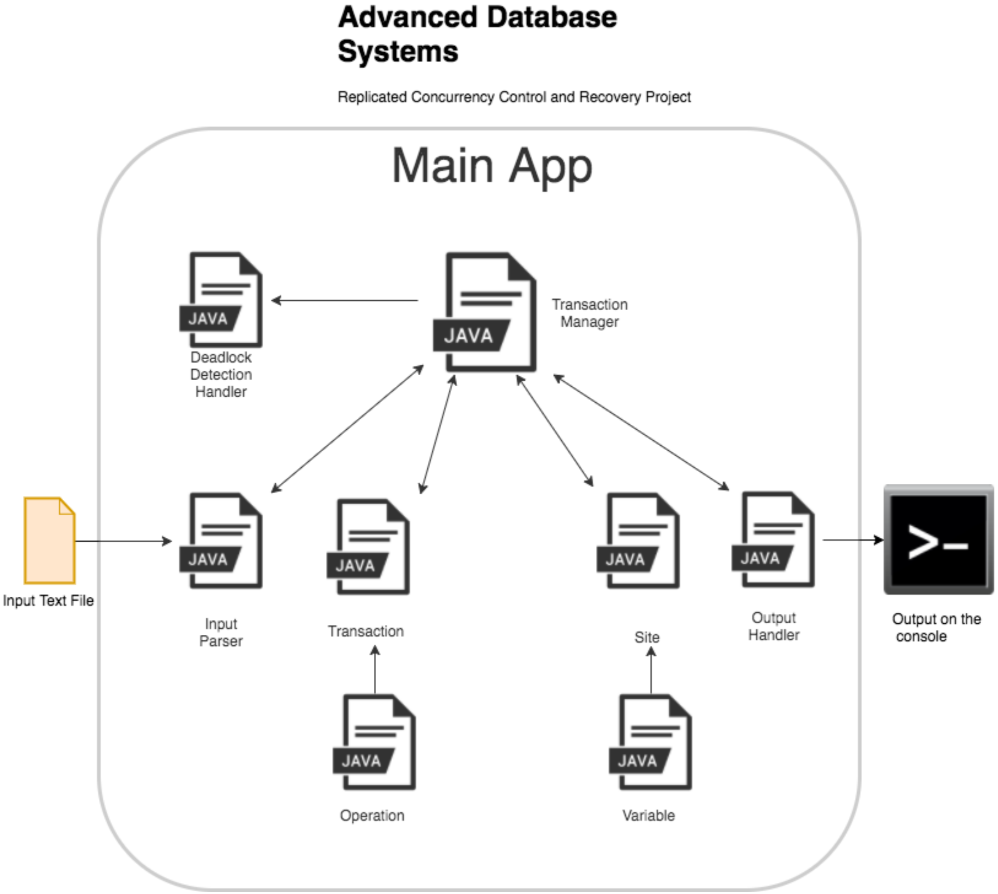


Figure 1: The output of the dump() command

5 Sample Test Runs

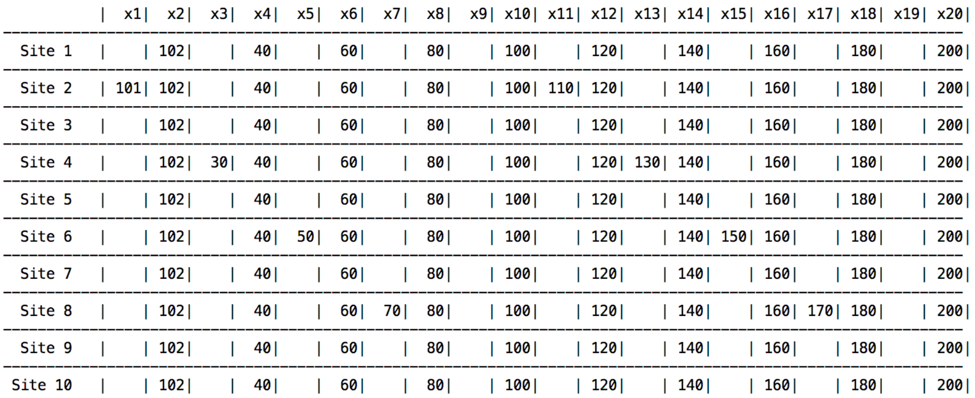
5.1 Test Case : 1(Basic Test case)

```
begin(T1)
begin(T2)
W(T1,x1,101)
W(T2,x2,202)
W(T1,x2,102)
W(T2,x1,201)
end(T1)
dump()
```

Following is the output of the above test case

```
Transaction{id=1} has begun!
Transaction{id=2} has begun!
```

```
Adding a write operation to Transaction{id=1}
Adding a write operation to Transaction{id=2}
Aborting Transaction{id=2} due to deadlock.
Adding a write operation to Transaction{id=1}
Transaction{id=1} committed successfully.
```



```
          |  x1|  x2|  x3|  x4|  x5|  x6|  x7|  x8|  x9| x10| x11| x12| x13| x14| x15| x16| x17| x18| x19| x20|
-----------------------------------------------------------------------------------------------------------------
  Site 1  |    | 102|    |  40|    |  60|    |  80|    | 100|    | 120|    | 140|    | 160|    | 180|    | 200|
-----------------------------------------------------------------------------------------------------------------
  Site 2  | 101| 102|    |  40|    |  60|    |  80|    | 100| 110| 120|    | 140|    | 160|    | 180|    | 200|
-----------------------------------------------------------------------------------------------------------------
  Site 3  |    | 102|    |  40|    |  60|    |  80|    | 100|    | 120|    | 140|    | 160|    | 180|    | 200|
-----------------------------------------------------------------------------------------------------------------
  Site 4  |    | 102|  30|  40|    |  60|    |  80|    | 100|    | 120| 130| 140|    | 160|    | 180|    | 200|
-----------------------------------------------------------------------------------------------------------------
  Site 5  |    | 102|    |  40|    |  60|    |  80|    | 100|    | 120|    | 140|    | 160|    | 180|    | 200|
-----------------------------------------------------------------------------------------------------------------
  Site 6  |    | 102|    |  40|  50|  60|    |  80|    | 100|    | 120|    | 140| 150| 160|    | 180|    | 200|
-----------------------------------------------------------------------------------------------------------------
  Site 7  |    | 102|    |  40|    |  60|    |  80|    | 100|    | 120|    | 140|    | 160|    | 180|    | 200|
-----------------------------------------------------------------------------------------------------------------
  Site 8  |    | 102|    |  40|    |  60|  70|  80|    | 100|    | 120|    | 140|    | 160| 170| 180|    | 200|
-----------------------------------------------------------------------------------------------------------------
  Site 9  |    | 102|    |  40|    |  60|    |  80|    | 100|    | 120|    | 140|    | 160|    | 180|    | 200|
-----------------------------------------------------------------------------------------------------------------
 Site 10  |    | 102|    |  40|    |  60|    |  80|    | 100|    | 120|    | 140|    | 160|    | 180|    | 200|
```

Figure 2: The output of the dump() command

5.2 Test Run : 2 (Read Only Transactions)

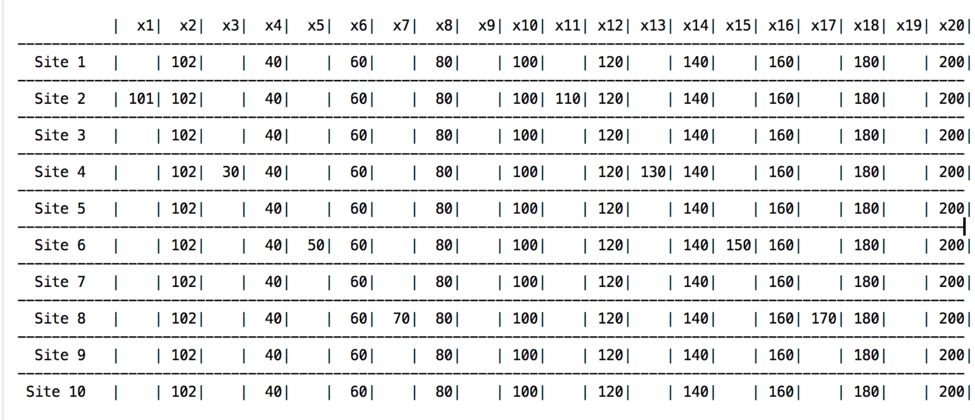
```
begin(T1)
beginRO(T2)
W(T1,x1,101)
R(T2,x2)
W(T1,x2,102)
R(T2,x1)
end(T1)
end(T2)
dump()
```

Following is the output:

```
Transaction{id=1} has begun!
Adding a buffered write operation to Transaction{id=1}
ReadOnlyTransaction{2} reads x2=20
Adding a buffered write operation to Transaction{id=1}
ReadOnlyTransaction{2} reads x1=10
Transaction{id=1} committed successfully.
```

5.3 Test Run : 3(Fail, recover with Circular deadlock scenario)

```
begin(T1)
```



```
          |  x1|  x2|  x3|  x4|  x5|  x6|  x7|  x8|  x9| x10| x11| x12| x13| x14| x15| x16| x17| x18| x19| x20|
----------------------------------------------------------------------------------------------------------------
  Site 1  |    | 102|    |  40|    |  60|    |  80|    | 100|    | 120|    | 140|    | 160|    | 180|    | 200|
----------------------------------------------------------------------------------------------------------------
  Site 2  | 101| 102|    |  40|    |  60|    |  80|    | 100| 110| 120|    | 140|    | 160|    | 180|    | 200|
----------------------------------------------------------------------------------------------------------------
  Site 3  |    | 102|    |  40|    |  60|    |  80|    | 100|    | 120|    | 140|    | 160|    | 180|    | 200|
----------------------------------------------------------------------------------------------------------------
  Site 4  |    | 102|  30|  40|    |  60|    |  80|    | 100|    | 120| 130| 140|    | 160|    | 180|    | 200|
----------------------------------------------------------------------------------------------------------------
  Site 5  |    | 102|    |  40|    |  60|    |  80|    | 100|    | 120|    | 140|    | 160|    | 180|    | 200|
----------------------------------------------------------------------------------------------------------------
  Site 6  |    | 102|    |  40|  50|  60|    |  80|    | 100|    | 120|    | 140| 150| 160|    | 180|    | 200|
----------------------------------------------------------------------------------------------------------------
  Site 7  |    | 102|    |  40|    |  60|    |  80|    | 100|    | 120|    | 140|    | 160|    | 180|    | 200|
----------------------------------------------------------------------------------------------------------------
  Site 8  |    | 102|    |  40|    |  60|  70|  80|    | 100|    | 120|    | 140|    | 160| 170| 180|    | 200|
----------------------------------------------------------------------------------------------------------------
  Site 9  |    | 102|    |  40|    |  60|    |  80|    | 100|    | 120|    | 140|    | 160|    | 180|    | 200|
----------------------------------------------------------------------------------------------------------------
 Site 10  |    | 102|    |  40|    |  60|    |  80|    | 100|    | 120|    | 140|    | 160|    | 180|    | 200|
```

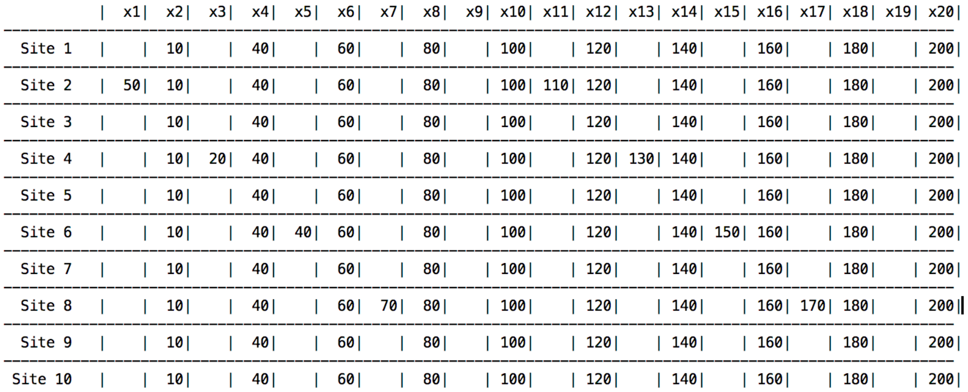
Figure 3: The output of the dump() command

```
begin(T2)
begin(T3)
begin(T4)
begin(T5)
R(T3,x3)
fail(4)
recover(4)
R(T4,x4)
R(T5,x5)
R(T1,x6)
R(T2,x2)
W(T1,x2,10)
W(T2,x3,20)
W(T3,x4,30)
W(T5,x1,50)
end(T5)
W(T4,x5,40)
end(T4)
end(T3)
end(T2)
end(T1)
```

Following is output of the sample test case above:

```
Transaction{id=1} has begun!
Transaction{id=2} has begun!
Transaction{id=3} has begun!
Transaction{id=4} has begun!
Transaction{id=5} has begun!
Adding a buffered read operation to Transaction{id=3}
Site{id=4} failed
```

```
Site{id=4} recovered
Adding a read operation to Transaction{id=4}
Adding a read operation to Transaction{id=5}
Adding a read operation to Transaction{id=1}
Adding a read operation to Transaction{id=2}
Adding a write operation to Transaction{id=2}
Adding a write operation to Transaction{id=5}
Transaction{id=5} read var x5=50 from site 6
Transaction{id=5} committed successfully.
Adding a buffered write operation to Transaction{id=4}
Transaction{id=4} read var x4=40 from site 1
Transaction{id=4} committed successfully.
Adding a buffered write operation to Transaction{id=3}
Aborting Transaction{id=3}. Commit Failed
Transaction{id=2} read var x2=20 from site 1
Transaction{id=2} committed successfully.
Adding a buffered write operation to Transaction{id=1}
Transaction{id=1} read var x6=60 from site 1
Transaction{id=1} committed successfully.
```



```
           |  x1|  x2|  x3|  x4|  x5|  x6|  x7|  x8|  x9| x10| x11| x12| x13| x14| x15| x16| x17| x18| x19| x20|
---------------------------------------------------------------------------------------------------------------
  Site 1   |    |  10|    |  40|    |  60|    |  80|    | 100|    | 120|    | 140|    | 160|    | 180|    | 200|
---------------------------------------------------------------------------------------------------------------
  Site 2   |  50|  10|    |  40|    |  60|    |  80|    | 100| 110| 120|    | 140|    | 160|    | 180|    | 200|
---------------------------------------------------------------------------------------------------------------
  Site 3   |    |  10|    |  40|    |  60|    |  80|    | 100|    | 120|    | 140|    | 160|    | 180|    | 200|
---------------------------------------------------------------------------------------------------------------
  Site 4   |    |  10|  20|  40|    |  60|    |  80|    | 100|    | 120| 130| 140|    | 160|    | 180|    | 200|
---------------------------------------------------------------------------------------------------------------
  Site 5   |    |  10|    |  40|    |  60|    |  80|    | 100|    | 120|    | 140|    | 160|    | 180|    | 200|
---------------------------------------------------------------------------------------------------------------
  Site 6   |    |  10|    |  40|  40|  60|    |  80|    | 100|    | 120|    | 140| 150| 160|    | 180|    | 200|
---------------------------------------------------------------------------------------------------------------
  Site 7   |    |  10|    |  40|    |  60|    |  80|    | 100|    | 120|    | 140|    | 160|    | 180|    | 200|
---------------------------------------------------------------------------------------------------------------
  Site 8   |    |  10|    |  40|    |  60|  70|  80|    | 100|    | 120|    | 140|    | 160| 170| 180|    | 200|
---------------------------------------------------------------------------------------------------------------
  Site 9   |    |  10|    |  40|    |  60|    |  80|    | 100|    | 120|    | 140|    | 160|    | 180|    | 200|
---------------------------------------------------------------------------------------------------------------
 Site 10   |    |  10|    |  40|    |  60|    |  80|    | 100|    | 120|    | 140|    | 160|    | 180|    | 200|
```

Figure 4: The output of the dump() command

6 Conclusions

Here you briefly summarize your findings.