UNIVERSITY OF TRENTO

Optimistic Concurrency Control for Distributed Transactions

Project
Distributed Systems 1 Course 2020/21

July 16, 2021

Contents

1 Structure of the project

In the folder src/main/java, under the package it.unitn.ds1, we have the following Actor classes

- Coordinator
- Server
- TxnClient

And the following non-Actor classes

- ConsistencyTester, is used to display the distributed data store status when the execution is terminated by the user.
- CtrlSystem, handles the initialization and termination of the actors.
- DataItem, represents a data item of the data store.
- DataOperation, represents an operation over a specific data item.
- Txn, represents a transaction.

The project relies on

- Gradle as a build tool.
- Apache Log4j v2 to manage the system output, its settings are collected in the configuration file log4j2.xml under the path src/resources.
- The usual Akka 2.6.13 to implement the actor model.

2 Main design choices

2.1 Binding operations to transactions

As previously said, we use the DataOperation class to represent operations and Txn class to represent transactions. Let's consider a single Coordinator instance

- Each TxnClient is granted one and only one transaction per time.
- Each Txn instance is stored in a Map object and used as a key of the map.
- Each DataOperation instance is added to a list of operations that is stored as a value of the map.

2.2 Private Workspace

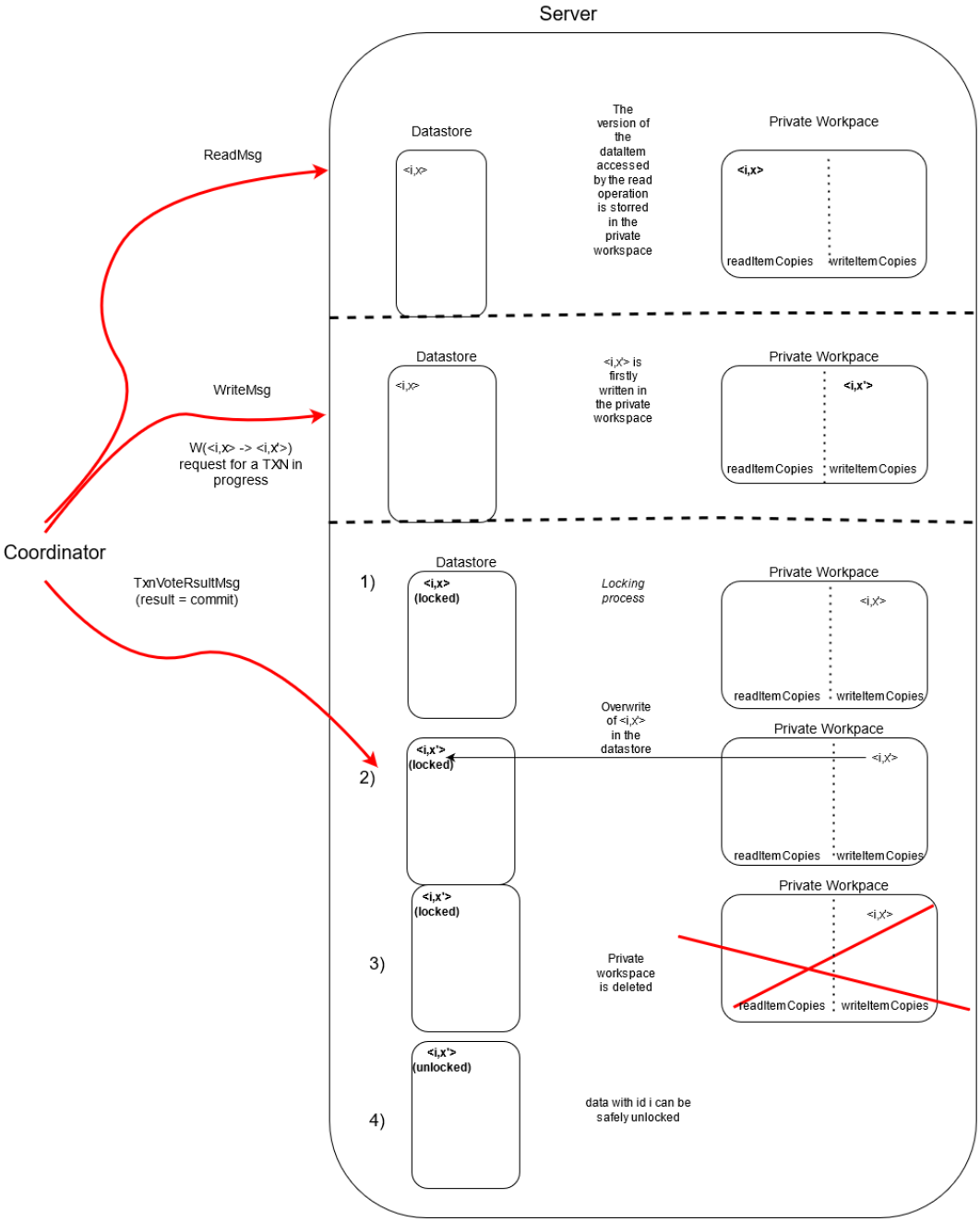
Private Workspaces are created by the server when he receives a read or write request from a new transaction. The private workspace contains two list of copies :

- Read Copies : contains copies of the data items accessed by a read request;
- Write Copies : contains copies of the overwriting data items of all write requests for a transaction.

So, the write request and read requests are always trying to access the Write Copies of the Private Workspace in order to retrieve the last version of the data item willing to be overwritten or read.

After the first request received by the server, the server will always try to access the Write Copies of the private workspace in order to retrieve the last version of the data item willing to be overwritten or read. Then the private workspace will contain a full list of the data items accessed in the transaction.

When the server receives the order to overwrite, it selects every data item in the Write Copies of the private workspace and edited directly the datastore in the order of the write operations requested during the transaction. Finally, the private workspace can be deleted and the data items involved in the transaction can be unlocked. The full operation of the private workspace is described below.



Because we are working in the optimistic scenarios the end of the transaction the private

workspace can contain inconsistent states, because a concurrent transaction might have changed the data store during the transaction. But the validation process will detect the inconsistency in that case and the decision will eventually be 'ABORT' for the transaction.

2.3 Validation and commit

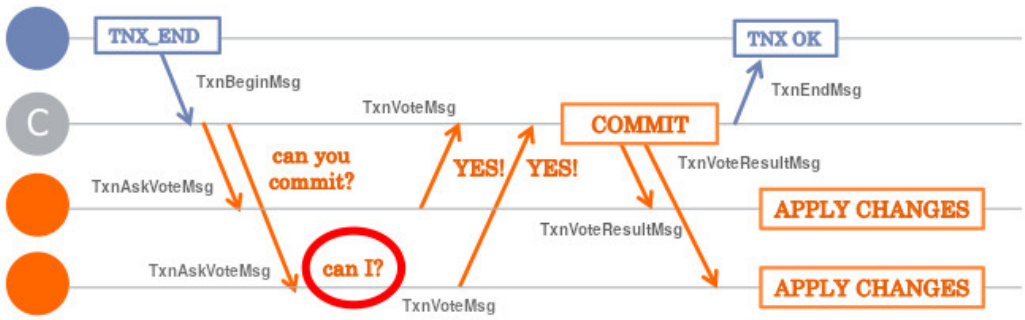
The validation phase starts as soon as a Coordinator instance receives a TXN-END message from a client holding an open transaction.

1. Coordinator retrieves the Server instances that are involved in the list of operations requested by the Txn instance. To these instances, Coordinator sends a TxnAskVoteMsg.
2. Upon receiving a TxnAskVoteMsg, the Server instance locks the items that are involved in the Txn contained in the message. As soon as these items are locked, the Server validates the private workspace against its data store.
3. If the validation is successful, the Server responds to the Coordinator with a COMMIT vote through a TxnVoteResultMsg and waits. An unsuccessful validation causes the Server to immediately abort and respond to the Coordinator with an ABORT vote.

So the Server instances that casted a COMMIT vote must wait the Coordinator confirmation before applying the changes. Once the Coordinator instance collects all the votes, we have two cases

1. There is no ABORT vote, so the Coordinator sends a TxnVoteResultMessage with a COMMIT vote to all the involved servers.
2. There is at least one ABORT vote, so the Coordinator sends a TxnVoteResultMessage with an ABORT vote to all the involved servers.

Finally, the TxnClient instance is informed of the result of its transaction.



3 Implementation

3.1 Assumptions

The assumptions include the one set by the assignment, so

- Links are FIFO and reliable.
- Unicast transmission delays are simulated with small, random intervals.
- Clients don't crash.

Additional assumptions are made for the current implementation

- Coordinators don't crash.
- Servers don't crash.

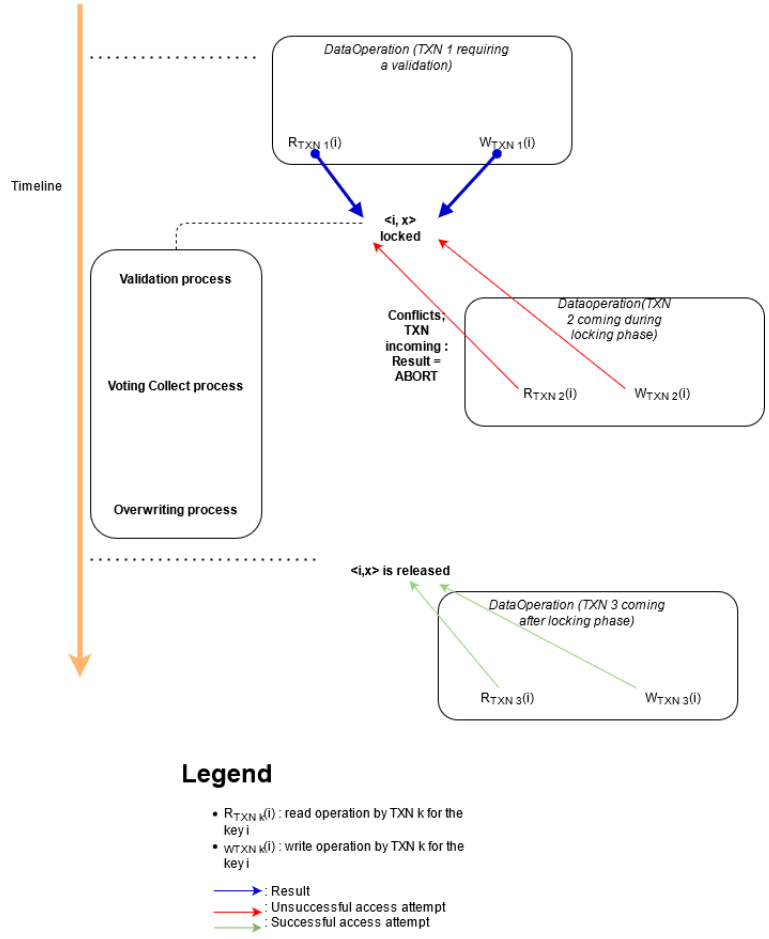
3.2 Safeness

In every server involved in a transaction, the validation checks in the private workspace that the copies following condition :

1. data item in Read Copies : the data item must be present in the datastore or in the 'Write copies' of the private workspace (version, value and the dataid must be the same);
2. data item in Write Copies : every copy must have a version superior than the data item with the same id in the datastore.

We have to ensure that no concurrent transaction attempts to access a data item during the validation and the application of the changes phase. So we implemented locks for the data items involved in the transaction. The locks are released after the changes are applied. If a concurrent transaction tries to access these data items for a read or write request during these phases : the decision must always be commit.

We describe this with the diagram below :



Legend

- $R_{TXN\ k}(i)$: read operation by TXN k for the key i
- $W_{TXN\ k}(i)$: write operation by TXN k for the key i

: Result
: Unsuccessful access attempt
: Successful access attempt

3.3 Improvements

There are some improvements that could have been implemented to represent a real system where actors and servers can fail. So the project could have benefited from these work.

- Implementing the failure of coordinators and servers.
- Implementing fault tolerance.

Furthermore we observed an unexpected behavior for a high number of servers (» 10 servers) where a null pointer error occurs, but with no threat to consistency.