

2-phase distributed commit (2PC) protocol

by

Divya Boggavarapu(1002086719)

Abhishek Reddy Yaramala(1002072996)

I have neither given nor received unauthorised assistance on this work.

Sign: Divya Boggavarapu

Date: 11/01/2022

Sign: Abhishek Reddy Yaramala

Date: 11/01/2022

Table of Contents

Table of Contents	2
Project Description:	3
Contribution:	3
Requirements	3
Java Socket	3
2PC protocol in Distributed Systems	4
How does two-phase commit work?	4
Implementation Steps:	6
Detailed Instructions to Execute Code	6
Screenshots:	7
What issues encountered	10
What we learnt	10

Project Description:

In this project a 2-phase distributed commit(2PC) protocol and use controlled and randomly injected failures to study how the 2PC protocol handles node crashes. Assumed one transaction coordinator (TC) and at least two participants in the 2PC protocol. Similar to the previous projects, we used multiple processes to emulate multiple nodes. Each node (both the TC and the participants) devises a time-out mechanism when no response is received and transits to either the abort or commit state. We assumed that only one node fails in the controlled test. Evaluate different possibilities of failures:

TC failure : If the coordinator fails before sending the "prepare" message, nodes will not receive the "prepare" message until the time-out and will abort. So, they will respond "no" to the "prepare" message after the coordinator comes back up and sends the "prepare" message.

Node failure : If the transaction coordinator does not receive "yes" from a node, it will abort the transaction.

Contribution:

This project is developed by Divya Boggavarapu(1002086719) and Abhishek Reddy Yaramala(1002072996).

Requirements

- Java
- Eclipse IDE

Java Socket

A socket is an endpoint for communication between two machines. The actual work of the socket is performed by an instance of the SocketImpl class. An application, by changing the socket factory that creates the socket implementation, can configure itself to create sockets appropriate to the local firewall.

java.net.Socket class.

```
Socket socket = new Socket( server, port );
```

2PC protocol in Distributed Systems

Two-phase commit (2PC) is a standardized protocol that ensures atomicity, consistency, isolation and durability ([ACID](#)) of a transaction; it is an [atomic](#) commitment protocol for distributed systems.

Transactions in a distributed system involve changing data on multiple databases or resource managers, making processing more challenging because the database must coordinate the committing or rolling back of changes in a transaction as a whole; either the whole transaction commits, or the whole transaction rolls back.

A transaction manager makes use of 2PC to maintain data integrity, the integrity of the global database (the group of databases involved in the transaction), and to keep track of the distributed transactions' commitment or rollback. The user or application developer need not know any programming to use this protocol because it is completely transparent.

How does two-phase commit work?

A coordinator is a unique object that is necessary for a distributed transaction to happen. The coordinator is responsible for organising interactions and synchronizations across distributed servers.

2PC, as its name suggests, has two phases:

Phase 1 (the prepare phase): The protocol verifies that the updates from the transaction have been stored to stable storage by each resource management. Every server that must commit logs the records of its data in a log. A server will respond with a failure message if it is unsuccessful in accomplishing this; an OK message if it is successful.

The starting node asks all other participating nodes to pledge to either commit or roll back the transaction during this initial phase.

The responding node has three options for returning responses:

Prepared - When data in the node has been changed by a statement in the distributed transaction and the node has successfully assembled itself for commitment or rollback, a prepared response is given. In addition, the planned reaction makes sure that locks held for the transaction can withstand a failure.

- Read-only When a node returns a read-only answer, data has been accessed but cannot be changed. As a result, no preparation is required.

Abort: If the node receives an abort response, it cannot properly become ready for commitment.

Each node, excluding the commit point site, must take a number of actions in order for the prepare phase to end and one of the three messages to be sent. The node must first ask if the nodes listed below are prepared to commit before proceeding. The node then determines if the transaction affects data on it or subsequent nodes. The node skips the remaining steps and responds with a read-only response if the data does not change.

The node assigns the resources required to commit the transaction if the data does change. The node will add redo records to its redo log that correspond to the modifications made by the transaction. The updated tables are then locked to prevent reading of them.

The node then makes sure that any locks held for the transaction are resilient to failure. If everything goes as planned, the node sends out a prepared response. The node will, however, send the abort answer if it, or one of its succeeding nodes, is unable to prepare to commit.

Then, prepared nodes watch for the global coordinator to commit or rollback its decision. Until all changes are either committed or undone, the prepared nodes are regarded as in-doubt.

Phase 2 (the commit phase) is where all resource managers are instructed to commit if phase one is a success and all participants respond with a "yes." Each node logs its commit in a record and notifies the coordinator that it was successful after committing by sending a message. If phase one is unsuccessful, step two instructs the resource managers to abort, rolls back all servers, and each node transmits feedback indicating the rollback was successful.

The following steps can be used to breakdown the commit phase:

- The action is carried out after the global coordinator instructs the commit point site to do so.
- The commit point site logs the commitment and replies to the global coordinator to let them know the commitment was successful.
- All other nodes are given instructions by the global and local coordinators to commit to the transaction.
- The databases on each node release their locks and finalize their local contributions to the distributed transaction.
- The local log of each node's database records an additional redo entry to indicate that the transaction has been committed.
- Each participating node notifies the global coordinator when a commitment is successful.

All nodes in the distributed system have consistent data after the commit step is finished.

Databases can independently fail and recover in a distributed system. Due to a system failure, a transaction may successfully commit its updates on one database system but fail to do so on another. The failing database must be capable of committing the transaction once it has recovered. To do this, the system needs to have a copy of the updates that were made there throughout the transaction. The information in a system's primary memory could be lost, though, if it malfunctions.

Therefore, before a failure happens, the database must save a copy of the updates made by the transaction. Before a transaction commits anywhere, 2PC makes sure that every system it accesses durably maintains its portion of the transaction's modifications.

Usually, a transaction manager implements 2PC. The transaction manager implements the 2PC protocol and keeps track of which resource managers are accessed by each transaction.

Implementation Steps:

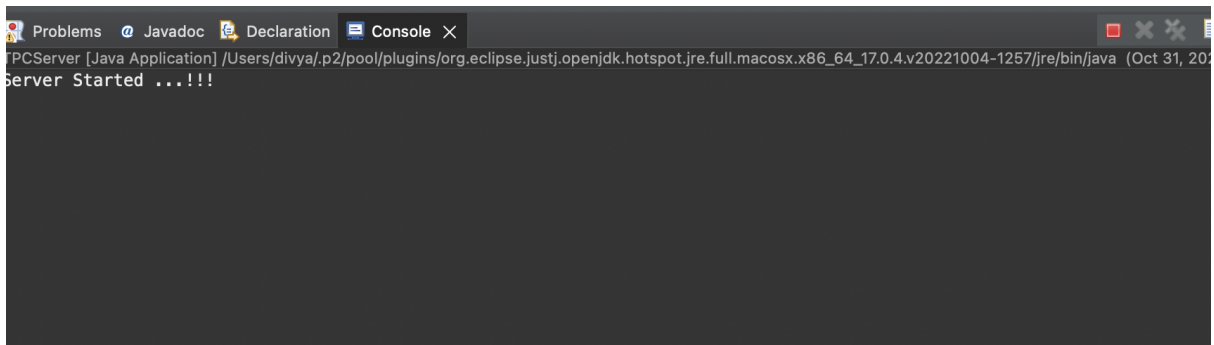
- Created three java classes with the operations of Server, Co-ordinator and the participants.
- Implemented TPCTServer.Java, started server socket on port 9800. Created threads for co-ordinator and participants.
- Implemented TPCCoordinator.Java where we created the coordinator thread and implemented various methods to check whether the participants voted or not.
- If Voted COMMIT by both participants it'll print Global_Commit. If ABORT by any of the participants it'll print Global_Abort.
- Implemented TPCParticipant.Java to create threads for participants and methods to COMMIT or ABORT messages from Coordinator.
- We also have set a 90 seconds time to ABORT the message if the Participant has not responded.

Detailed Instructions to Execute Code

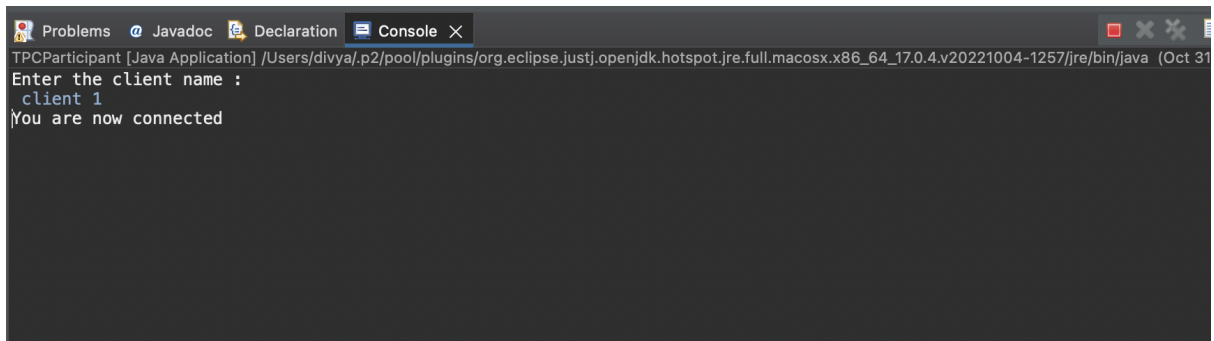
- Run TPCTServer.Java, TPCCoordinator.Java and run TPCParticipant.Java two times.
- Four java processes will start running
- We need to register our participants by entering the name of participants.
- We will get messages to the Coordinator that our participants are registered.
- Coordinator will ask if we want to send a message to our participants.

- We can switch to participants' console windows and choose to Commit or Abort the message.
- If we commit a message to all the participants, it's a Global Commit.
- If we Aborted a message in either one or two it's a Global Abort.
- Even if one of the participants didn't respond for Particular time, it'll be considered as Abort.
- All these operations will be logging on the server.

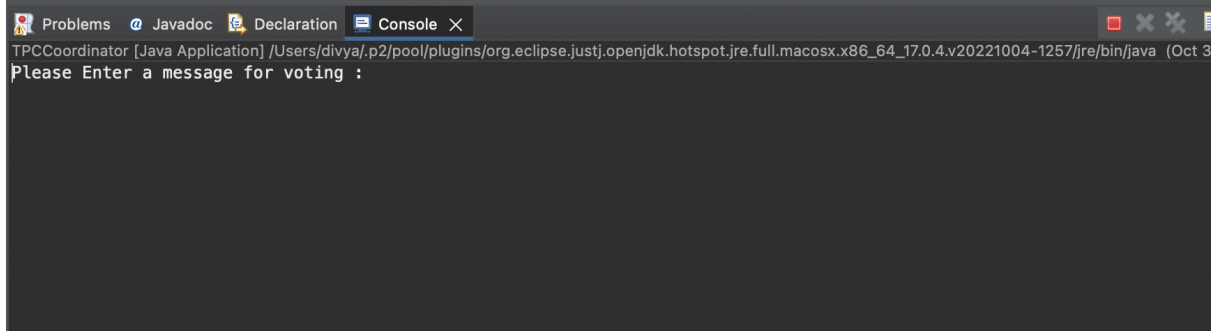
Screenshots:



```
TPCServer [Java Application] /Users/divya/.p2/pool/plugins/org.eclipse.justj.openjdk.hotspot.jre.full.macosx.x86_64_17.0.4.v20221004-1257/jre/bin/java (Oct 31, 2022 10:04:12 AM)  
Server Started ....!!!
```



```
TPCParticipant [Java Application] /Users/divya/.p2/pool/plugins/org.eclipse.justj.openjdk.hotspot.jre.full.macosx.x86_64_17.0.4.v20221004-1257/jre/bin/java (Oct 31, 2022 10:04:12 AM)  
Enter the client name :  
client 1  
You are now connected
```



```
TPCCoordinator [Java Application] /Users/divya/.p2/pool/plugins/org.eclipse.justj.openjdk.hotspot.jre.full.macosx.x86_64_17.0.4.v20221004-1257/jre/bin/java (Oct 31, 2022 10:04:12 AM)  
Please Enter a message for voting :
```

```
Problems Javadoc Declaration Console X
TPCParticipant [Java Application] /Users/divya/.p2/pool/plugins/org.eclipse.justj.openjdk.hotspot.jre.full.macosx.x86_64_17.0.4.v20221004-1257/jre/bin/java (Oct 31
Enter the client name :
client2
You are now connected
```

```
Problems Javadoc Declaration Console X
TPCParticipant [Java Application] /Users/divya/.p2/pool/plugins/org.eclipse.justj.openjdk.hotspot.jre.full.macosx.x86_64_17.0.4.v20221004-1257/jre/bin/java (Oct 31
Enter the client name :
client 1
You are now connected
No Voting request received, Local Abort...!

Give a VOTE to COMMIT OR ABORT the message : hello both

COMMIT
currentState :: READY
Received global commit, so committing the transaction
currentState :: COMMIT
```

```
Problems Javadoc Declaration Console X
TPCParticipant [Java Application] /Users/divya/.p2/pool/plugins/org.eclipse.justj.openjdk.hotspot.jre.full.macosx.x86_64_17.0.4.v20221004-1257/jre/bin/java (Oct 31
Enter the client name :
client2
You are now connected
Give a VOTE to COMMIT OR ABORT the message : hello both

COMMIT
currentState :: READY
Received global commit, so committing the transaction
currentState :: COMMIT
```

```
Problems Javadoc Declaration Console X
TPCCoordinator [Java Application] /Users/divya/.p2/pool/plugins/org.eclipse.justj.openjdk.hotspot.jre.full.macosx.x86_64_17.0.4.v20221004-1257/jre/bin/java (Oct 3
Please Enter a message for voting :
A New participant : client 1 registered

A New participant : client2 registered

hello both
Voted 'COMMIT' by the client: client2

Voted 'COMMIT' by the client: client 1

All the participant clients voted to commit, so initiating global commit.
```



```
Problems Javadoc Declaration Console X
TPCServer [Java Application] /Users/divya/.p2/pool/plugins/org.eclipse.justj.openjdk.hotspot.jre.full.macosx.x86_64_17.0.4.v20221004-1257/jre/bin/java (Oct 31, 2022)
Server Started ...!!!
strLine ::: STARTED::COORDINATOR::TWOPC_COORDINATOR
strLine ::: STARTED::CLIENT:: client 1
strLine ::: STARTED::CLIENT::client2
strLine ::: hello both::VOTING_REQUEST
strLine ::: COMMIT
strLine ::: COMMIT
strLine ::: GLOBAL_COMMIT
```

```
Problems Javadoc Declaration Console X
TPCCoordinator [Java Application] /Users/divya/.p2/pool/plugins/org.eclipse.justj.openjdk.hotspot.jre.full.macosx.x86_64_17.0.4.v20221004-1257/jre/bin/java (Oct 31, 2022)
Please Enter a message for voting :
A New participant : client 1 registered

A New participant : client 2 registered

hello
Voted 'ABORT' by the client: client 1

Initiating the GLOBAL ABORT !

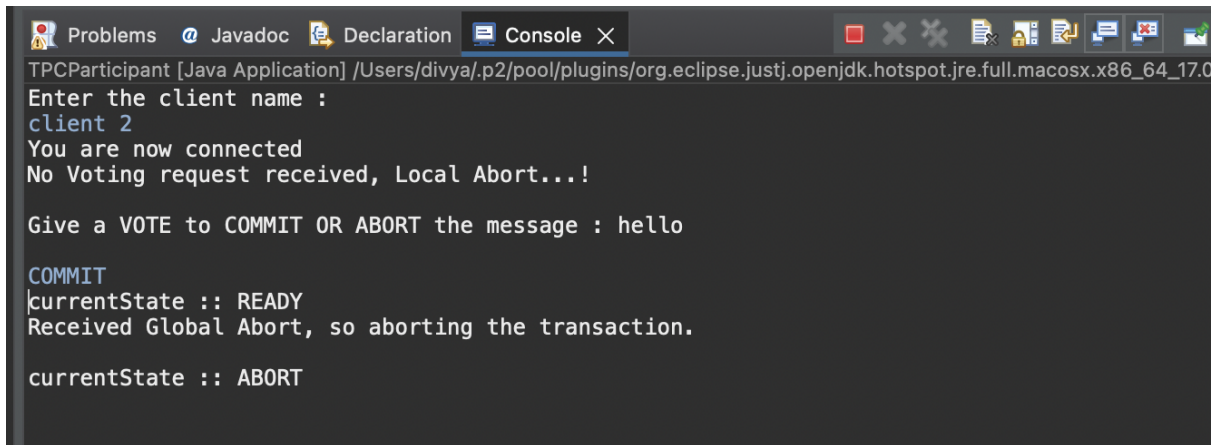
Voted 'COMMIT' by the client: client 2
```

```
Problems Javadoc Declaration Console X
TPCParticipant [Java Application] /Users/divya/.p2/pool/plugins/org.eclipse.justj.openjdk.hotspot.jre.full.macosx.x86_64_17.0.4.v20221004-1257/jre/bin/java (Oct 31, 2022)
Enter the client name :
client 1
You are now connected
No Voting request received, Local Abort...!

Give a VOTE to COMMIT OR ABORT the message : hello

ABORT
currentState :: ABORT
Received Global Abort, so aborting the transaction.

currentState :: ABORT
```



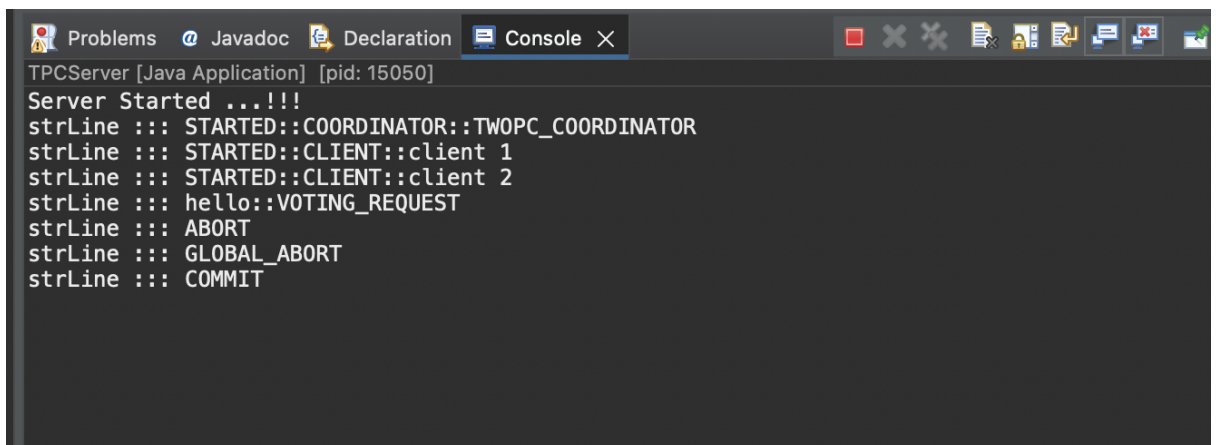
The screenshot shows the Eclipse IDE's console window for a Java application named 'TPCParticipant'. The user has entered 'client 2' as the client name. The application outputs 'You are now connected' and 'No Voting request received, Local Abort...!'. It then prompts for a vote on the message 'hello', and the user enters 'COMMIT'. The application then shows 'currentState :: READY' and 'Received Global Abort, so aborting the transaction.', followed by 'currentState :: ABORT'.

```
TPCParticipant [Java Application] /Users/divya/.p2/pool/plugins/org.eclipse.justj.openjdk.hotspot.jre.full.macosx.x86_64_17.0
Enter the client name :
client 2
You are now connected
No Voting request received, Local Abort...!

Give a VOTE to COMMIT OR ABORT the message : hello

COMMIT
currentState :: READY
Received Global Abort, so aborting the transaction.

currentState :: ABORT
```



The screenshot shows the Eclipse IDE's console window for a Java application named 'TPCServer'. The server has started successfully. It outputs a series of log messages: 'Server Started ...!!!', 'strLine ::: STARTED::COORDINATOR::TWOPC_COORDINATOR', 'strLine ::: STARTED::CLIENT::client 1', 'strLine ::: STARTED::CLIENT::client 2', 'strLine ::: hello::VOTING_REQUEST', 'strLine ::: ABORT', 'strLine ::: GLOBAL_ABORT', and 'strLine ::: COMMIT'.

```
TPCServer [Java Application] [pid: 15050]
Server Started ...!!!
strLine ::: STARTED::COORDINATOR::TWOPC_COORDINATOR
strLine ::: STARTED::CLIENT::client 1
strLine ::: STARTED::CLIENT::client 2
strLine ::: hello::VOTING_REQUEST
strLine ::: ABORT
strLine ::: GLOBAL_ABORT
strLine ::: COMMIT
```

What issues encountered

- Took some time to understand 2PC protocol and the Question.

What we learnt

- Got to learn about 2PC protocol in distributed systems.

