**Practical No. 6**

**Title: 2 PC Protocol Implementation**

**Aim: Implement Two Phase Commit (2PC) Protocol using Client Server Technology and Execute multiple Clients and a Server**

**Software required: MYSQL database, python**

**Theory:**

Two-phase commit (2PC) is a standardized protocol that ensures atomicity, consistency, isolation and durability ([ACID](https://searchsqlserver.techtarget.com/definition/ACID)) of a transaction; it is an [atomic](https://www.techtarget.com/whatis/definition/atomic) commitment protocol for distributed systems.

In a distributed system, [transactions](https://www.techtarget.com/searchcio/definition/transaction) involve altering data on multiple [databases](https://searchsqlserver.techtarget.com/definition/database) or resource managers, causing the processing to be more complicated since the database has to coordinate the committing or rolling back of changes in a transaction as a self-contained unit; either the entire transaction commits or the entire transaction rolls back.

**Some points to be considered regarding this protocol:**

**a)**In a two-phase commit, we assume that each site logs actions at that site, but there is no global log.

**b)**The **coordinator(Ci),**plays a vital role in doing confirmation whether the distributed transaction would abort or commit.

**c)** In this **protocol** messages are made to send between the **coordinator(Ci)** and the other **sites.** As each message is sent, its logs are noted at each sending site, to aid in recovery should it be necessary.

**The two phases of this protocol are as follow:**

**Phase-1st–**

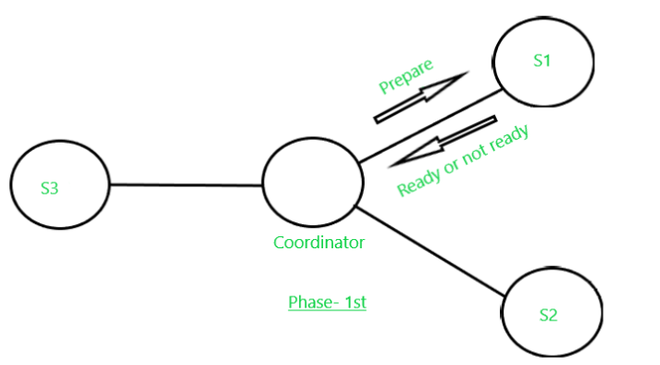
**a) Firstly, the coordinator(Ci)** places a log record **<Prepare T>** on the log record at its site.

**b)** Then, the **coordinator(Ci)** sends a **Prepare T**message to all the sites where the transaction(T) executed.

**c)** Transaction manager at each site on receiving this message **Prepare T** decides whether to commit or abort its component(portion) of T. The site can delay if the component has not yet completed its activity, but must eventually send a response.

**d)**If the site doesn’t want to commit, so it must write on **log record <no T>,**and local Transaction manager sends a message **abort T** to **Ci.**

**e)**If the site wants to commit, it must write on log record **<ready T>**, and local Transaction manager sends a message **ready T** to **Ci. Once the**ready**T message at Ciis sent**nothing can prevent it to commit its portion of **transaction T** except **Coordinator(Ci).**

. 

**Phase- 2nd–**

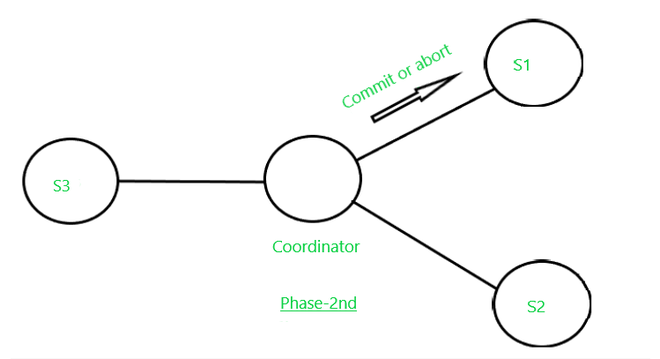
**The Second phase started as the response abort T or commit T receives by**the **coordinator(Ci) from all the sites that**are collaboratively **executing the transaction T.**However, it is possible that some site fails to respond; it may be down, or it has been disconnected by the network.  In that case, after a suitable timeout period will be given, after that time it will treat the site as if it had sent **abort T**. The fate of the transaction depends upon the following points:

a)  If the coordinator receives **ready T**from all the participating sites of T, then it decides to **commit T**. Then, the coordinator writes on its site log record **<Commit T>**  and sends a message **commit T** to all the sites involved in T.

b) If a site receives a **commit T** message, it commits the component of T at that site, and write it in log records **<Commit T>**.

c) If a site receives the message **abort T**, it aborts T and writes the log record **<Abort T>.**

d) However, if the coordinator has received **abort T**from one or more sites, it logs **<Abort T>** at its site and then sends **abort T** messages to all sites involved in transaction T.



**Disadvantages:**

The major disadvantage of the Two-phase commit protocol is faced when the Coordinator site failure may result in blocking, so a decision either to commit or abort **Transaction(T)** may have to be postponed until coordinator recovers.

**Source Code :-**

**Server Implementation:**

import socket

import threading

class TwoPhaseCommitServer:

def \_\_init\_\_(self, host, port):

self.host = host

self.port = port

self.participants = []

self.decisions = {}

def start\_server(self):

with socket.socket(socket.AF\_INET, socket.SOCK\_STREAM) as server\_socket:

server\_socket.bind((self.host, self.port))

server\_socket.listen()

print(f"Server listening on {self.host}:{self.port}")

while True:

client\_socket, client\_address = server\_socket.accept()

threading.Thread(target=self.handle\_client, args=(client\_socket,)).start()

def handle\_client(self, client\_socket):

data = client\_socket.recv(1024).decode('utf-8')

if data.startswith("REGISTER"):

participant\_id = int(data.split(" ")[1])

self.participants.append(client\_socket)

self.decisions[participant\_id] = None

print(f"Participant {participant\_id} registered.")

elif data.startswith("VOTE"):

participant\_id, decision = map(int, data.split(" ")[1:])

self.decisions[participant\_id] = decision

print(f"Participant {participant\_id} voted {decision}.")

client\_socket.close()

def send\_decision\_request(self):

decision\_request = "DECIDE"

for participant in self.participants:

participant.sendall(decision\_request.encode('utf-8'))

print("Decision request sent to all participants.")

# Wait for decisions from participants

while None in self.decisions.values():

pass

# Make the final decision based on participant votes

final\_decision = all(x == 'COMMIT' for x in self.decisions.values())

if final\_decision:

print("Global decision: COMMIT")

else:

print("Global decision: ABORT")

if \_\_name\_\_ == "\_\_main\_\_":

server = TwoPhaseCommitServer('127.0.0.1', 8888)

server\_thread = threading.Thread(target=server.start\_server)

server\_thread.start()

# Wait for participants to register

input("Press Enter to start the 2PC protocol...")

# Send decision request to participants

server.send\_decision\_request()

**Client Implementation:**

import socket

class TwoPhaseCommitClient:

def \_\_init\_\_(self, host, port, participant\_id):

self.host = host

self.port = port

self.participant\_id = participant\_id

def register(self):

with socket.socket(socket.AF\_INET, socket.SOCK\_STREAM) as client\_socket:

client\_socket.connect((self.host, self.port))

registration\_message = f"REGISTER {self.participant\_id}"

client\_socket.sendall(registration\_message.encode('utf-8'))

def vote(self, decision):

with socket.socket(socket.AF\_INET, socket.SOCK\_STREAM) as client\_socket:

client\_socket.connect((self.host, self.port))

vote\_message = f"VOTE {self.participant\_id} {decision}"

client\_socket.sendall(vote\_message.encode('utf-8'))

if \_\_name\_\_ == "\_\_main\_\_":

participant\_id = int(input("Enter participant ID: "))

client = TwoPhaseCommitClient('127.0.0.1', 8888, participant\_id)

# Register participant

client.register()

# Participant votes

decision = input("Vote (COMMIT/ABORT): ")

client.vote(decision)

**Conclusion :-**

We have implemented Two Phase Commit (2PC) Protocol using Client Server Technology

**FAQ -**

### Explain Key Characteristics Of Distributed Systems

### What Is A Distributed Lock And Why Is It Important

### What is 2 phase commit protocol and explain how site failure is handled?

## What is two-phase commit in the database

### explain the concept of savepoints in database transactions?

### 

### 