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) of a transaction; it is an atomic commitment protocol for distributed systems.

In a distributed system, transactions involve altering data on multiple databases 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(C_i),** 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(C_i)** 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–

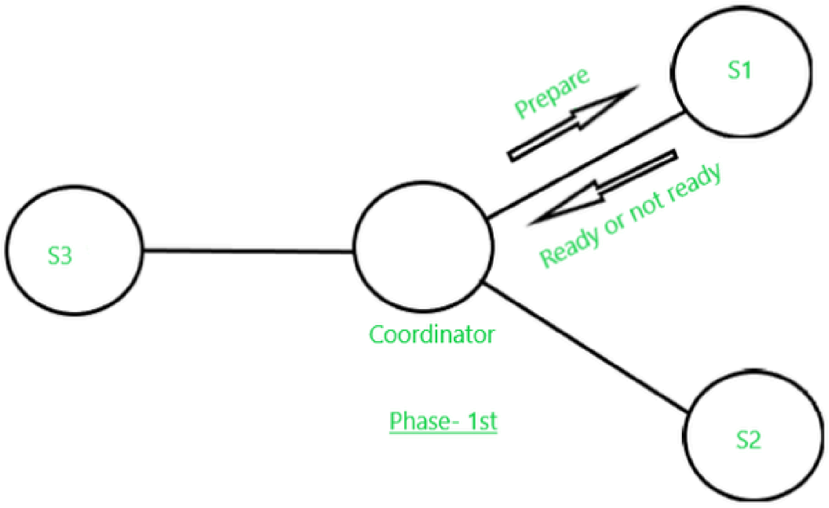
a) Firstly, the coordinator(C_i) places a log record **<Prepare T>** on the log record at its site.

b) Then, the **coordinator(C_i)** 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 **C_i.**

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 **C_i. Once the** ready **T message at C_i is sent** nothing can prevent it to commit its portion of **transaction T** except **Coordinator(C_i).**



.

Phase- 2nd–

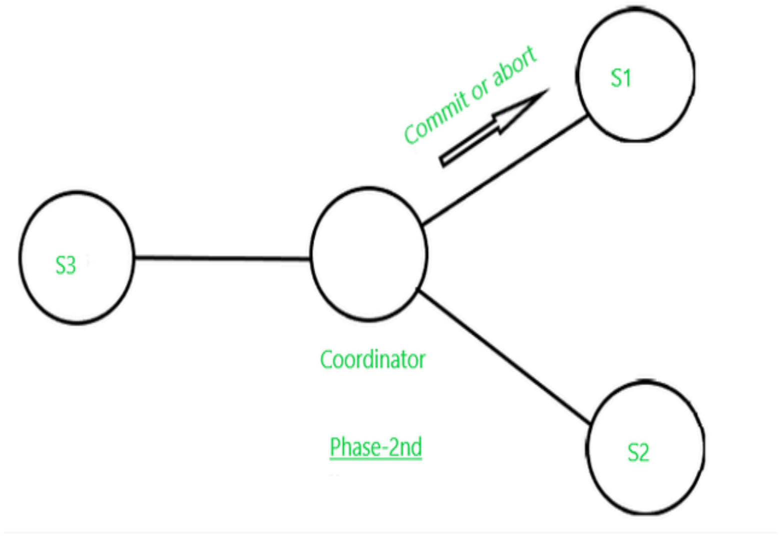
The Second phase started as the response abort T or commit T receives by the **coordinator(C_i) from all the sites that** are collaboratively **executing the transaction T.** However, it is possible that some site fails to respond; it may be down, or it has been disconnected by the network. In that case, 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 -

1) Explain Key Characteristics Of Distributed Systems
2) What Is A Distributed Lock And Why Is It Important
3) What is 2 phase commit protocol and explain how site failure is handled?
4) What is two-phase commit in the database

5) explain the concept of savepoints in database transactions?