GROUP PROJECT

Team Name: Cluster Crafters (Mobile Phone Supply Chain Management)

PART 4: Distributed Transaction Management

This report analyzes the implementation of a Distributed Transaction Management system, this is the 4th part of our project.. The primary goals of this part is to develop mechanisms for handling transactions in a distributed environment, ensuring security and correctness of transactions across different locations, and evaluating the performance of transaction handling under various scenarios, including simultaneous operations and failure conditions.

The implementation includes 2 Python classes: ConcurrencyControl and RetryMechanism, run by a main class Task4. These components work together to manage transactions involving inventory operations in a database, presumably distributed.

Key Components:

ConcurrencyControl: Handles operations related to inventory management, including reducing inventory quantities and verifying the final state of the inventory. This class ensures that concurrent operations on the inventory data are managed correctly.

RetryMechanism: Implements a retry mechanism for transactions, utilizing the retry decorator from the retry module. This mechanism is crucial for maintaining the robustness of the system in the face of transient failures or operational anomalies.

Task4: Serves as the main class orchestrating the execution. It initializes the ConcurrencyControl and RetryMechanism with the necessary database connection string, and coordinates the process flow.

Implementation Details:

Transaction Handling: The ConcurrencyControl class manages database transactions involving inventory updates. It ensures that inventory quantities are reduced correctly and verifies the post-transaction state against the pre-transaction state.

Concurrency Management: The implementation utilizes Python's threading module to simulate concurrent transactions. Two threads are created and started to perform inventory reduction operations simultaneously.

Retry Logic: The RetryMechanism class demonstrates a retry mechanism where a transaction is attempted up to three times with a delay of two seconds between attempts. This is particularly useful in handling transient errors or temporary connectivity issues.

Database Interaction: PostgreSQL is used as the database system, with connections managed using the psycopg2 library. The implementation relies on environment variables for database connection configuration.

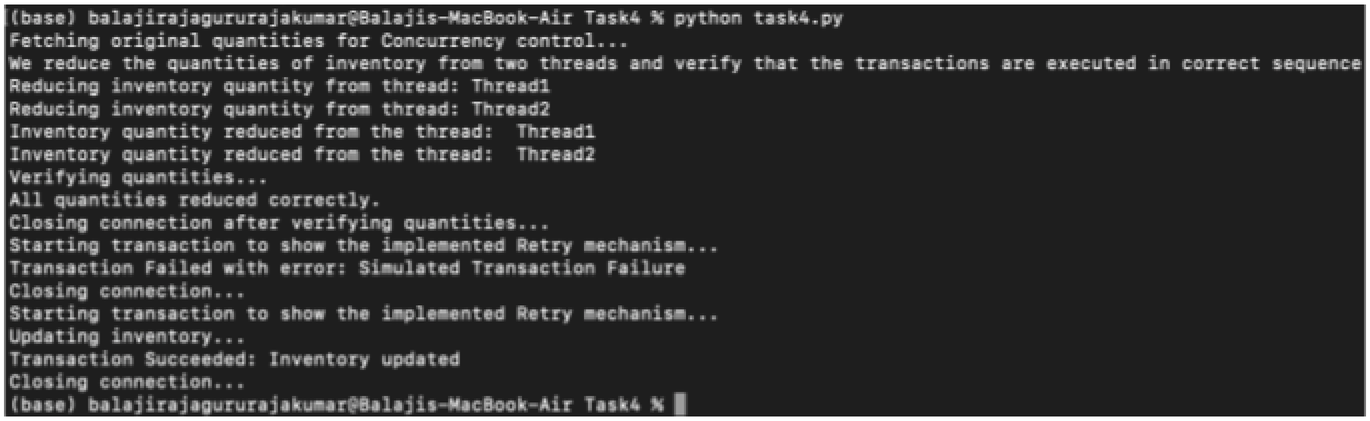
Error Handling and Logging: Both the ConcurrencyControl and RetryMechanism classes include error handling and logging, ensuring that transaction failures and operational errors are appropriately reported and managed.

Evaluation and Testing

Concurrency Testing: The concurrency aspect is tested by running two threads performing the same inventory reduction operation. The final quantities are verified against expected values to ensure correct concurrent behavior. This can be seen in the output screenshot below.

Retry Mechanism Testing: The retry mechanism is tested by simulating a failure in the first transaction attempt. Subsequent attempts and their outcomes demonstrate the effectiveness of the retry logic. This has been shown in the screenshot below.

Robustness and Security: The implementation includes basic error handling and transaction rollback mechanisms, contributing to the system's robustness. However, additional measures such as input validation, secure handling of environment variables, and detailed error logging would further enhance its security and reliability.



```
(base) balajirajagururajakumar@Balajis-MacBook-Air Task4 % python task4.py
Fetching original quantities for Concurrency control...
We reduce the quantities of inventory from two threads and verify that the transactions are executed in correct sequence
Reducing inventory quantity from thread: Thread1
Reducing inventory quantity from thread: Thread2
Inventory quantity reduced from the thread:  Thread1
Inventory quantity reduced from the thread:  Thread2
Verifying quantities...
All quantities reduced correctly.
Closing connection after verifying quantities...
Starting transaction to show the implemented Retry mechanism...
Transaction Failed with error: Simulated Transaction Failure
Closing connection...
Starting transaction to show the implemented Retry mechanism...
Updating inventory...
Transaction Succeeded: Inventory updated
Closing connection...
(base) balajirajagururajakumar@Balajis-MacBook-Air Task4 %
```

Figure 1: Output screenshot for Part 4

The implemented Distributed Transaction Management system effectively demonstrates mechanisms for handling transactions in a distributed database environment, with a focus on concurrency control and a retry mechanism for improved robustness. While the current implementation serves as a functional prototype, further enhancements in error handling, security features, and scalability considerations would be beneficial for real-world applications. The system's performance and reliability under various operational scenarios and failure conditions would also be an area for future exploration and testing.