MANIPAL INSTITUTE OF TECHNOLOGY
MANIPAL
(A constituent unit of MAHE, Manipal)

OPERATING SYSTEMS LAB (CSE)

MINI PROJECT REPORT:

Deadlock Detection in Database Transactions

(Project Title)

Names:
Anurag Nayak, 210905016, anurag.nayak@learner.manipal.edu
Adruti Onam, 210905190, adruti.onam@learner.manipal.edu
Moksha Kothari, 210905017, moksha.kothari@learner.manipal.edu
Vaibhhav Sandhir, 210915152, vaibhav.sandhir@learner.manipal.edu

Department of Computer Science and Engineering
Manipal Institute of Technology, Manipal.
November 2023

Abstract

This project investigates the phenomenon of deadlocks within database transactions, focusing on the detection and resolution mechanisms. It explores the underlying causes, algorithmic principles, and the practical applications of deadlock detection algorithms.

By simulating the detection process, the project provides an insight into the resource allocation graph's role in managing deadlocks. The study further analyses strategies employed by real-world database management systems and presents case studies to highlight the consequences and solutions to deadlocks.

The outcomes include a comprehensive report and a simulation model, contributing to a deeper understanding of deadlock management in contemporary database systems.

Problem Statement and Its Description

Introduction to the Problem:
Deadlocks in database systems are a critical issue where multiple transactions are stuck indefinitely, waiting for each other to release resource locks. This leads to operational halts and can severely impact the performance and integrity of a database management system.

Nature of Deadlocks:
Deadlocks arise from the concurrent execution of transactions, which is essential for DBMS efficiency and scalability. The Coffman conditions that lead to deadlocks include mutual exclusion, hold and wait, no preemption, and circular wait. A thorough understanding of these conditions is vital for developing effective deadlock management strategies.

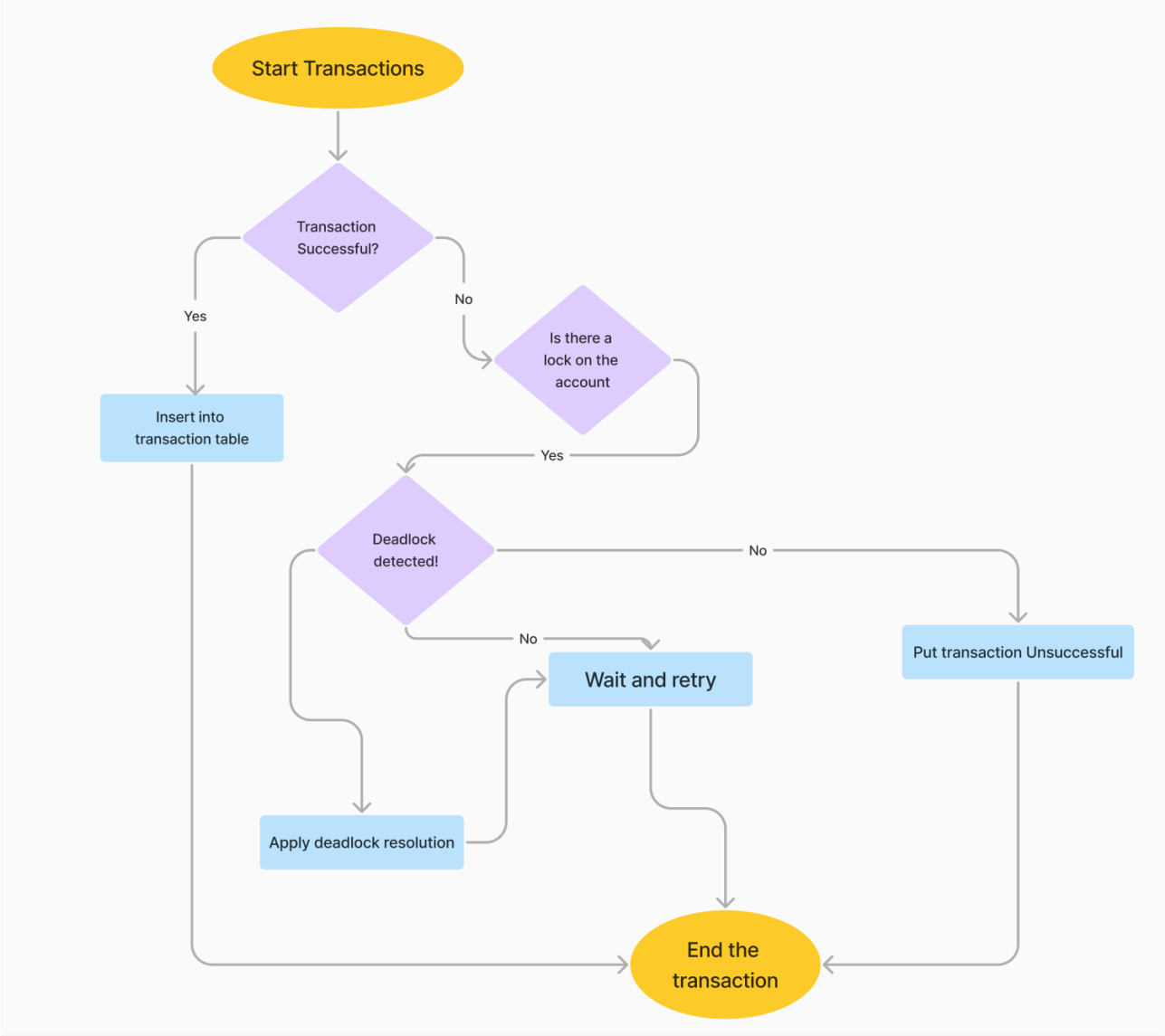
Impact on Database Operations:
A deadlock causes transactions to hold system resources indefinitely, affecting not only the transactions involved but potentially the entire system. The impact ranges from minor delays to severe disruptions, highlighting the need for prompt deadlock detection and resolution.

Deadlock Detection and Resolution:
Deadlock detection involves identifying cycles of dependencies, using algorithms like wait-for graphs or timeout mechanisms. Resolution strategies may include aborting transactions, rolling them back, or preempting resources to break the deadlock cycle.

Importance of Effective Management:
Effective deadlock management ensures the reliability and efficiency of a DBMS. It encompasses prevention, detection, and resolution of deadlocks through transaction design, database tuning, and concurrency control protocols, aiming to minimise deadlocks while maintaining transaction concurrency benefits.

Objectives of the Project:
The project aims to analyse the causes of deadlocks, study detection and resolution techniques, simulate deadlocks to evaluate their impact, and develop best practices for preventing deadlocks. The ultimate goal is to enhance the robustness and efficiency of DBMSs in handling deadlocks.

Algorithms/Flowcharts

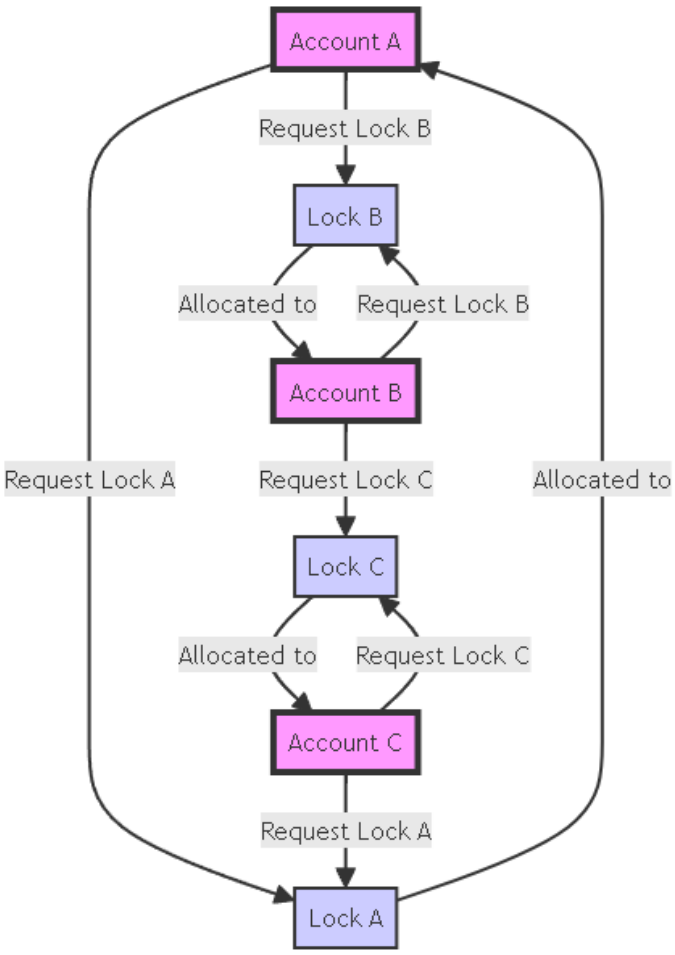


Working of the Code Explained

Algorithm: Transaction Handling with Deadlock Detection

1. Start Transaction
2. Attempt to execute the transaction
   - If the transaction is successful:
     a. Insert the transaction details into the transaction table in SQL
     b. End the transaction process
   - If the transaction is not successful:
     a. Check if there is a lock on the account
        - If there is a lock:
          i. Wait for a predefined time interval
          ii. Retry the transaction
        - If there is no lock:
          i. Check for deadlocks using cycle detection and the Wait-for graph algorithm
             - If a deadlock is detected:
               A. Apply deadlock resolution strategies (e.g., release locks)
               B. After resolution, wait for a predefined time interval
               C. Retry the transaction
             - If no deadlock is detected:
               A. Log the transaction failure
3. End of Transaction Process

Graphs :



Methodology

1. Literature Review:
Examination of existing research and techniques in deadlock detection.

2. Simulation Design:
Development of a simulation model to emulate deadlocks in a controlled environment of python and sql.

3. Algorithm Implementation:
Within the simulation environment, various deadlock detection algorithms will be implemented. These will include, but not be limited to, the Wait-for graph algorithm and cycle detection techniques as indicated in the flowchart. The algorithms will be tested for their ability to detect deadlocks accurately and efficiently. Additionally, the implementation will include mechanisms for resolving deadlocks once detected, such as transaction rollback or resource preemption.

4. Case Studies:
The project will analyse case studies of real-world deadlock incidents in database systems. These case studies will provide insights into how deadlocks have impacted database operations and the effectiveness of different resolution strategies that were employed. The lessons learned from these case studies will be used to inform the design of the simulation and the selection of algorithms.

5. Data Collection and Analysis:
During the simulation runs, data will be collected on various metrics such as the frequency of deadlocks, the time taken to detect and resolve deadlocks, and the impact on transaction throughput and system performance.This data will be analyzed to evaluate the effectiveness of the implemented deadlock detection and resolution algorithms. The analysis will help in identifying the strengths and weaknesses of each algorithm and in determining the best practices for managing deadlocks in database systems.

Implementation

1. Programming Languages and Tools:

- Python used for its extensive libraries and MySQL for database management.

- MySQL connector in Python facilitated database interactions.

2. Database Schema and Transaction Scenarios:

- Created `DeadlockDemo` database with `Accounts` and `Transactions` tables.

- `Accounts` table included fields for `id`, `name`, and `balance`.

- `Transactions` table tracked transfers with fields for `transaction_id`, `src_id`, `dest_id`, `amount`, and `transaction_time`.

3. Simulation Environment Setup:

- Utilised Visual Studio Code (VS Code) as the IDE for development.

- Implemented version control with Git and GitHub for collaborative work.

4. Transaction Execution and Deadlock Induction:

- Python scripts executed transactions using MySQL cursors.

- Simulated concurrent transactions to induce potential deadlocks.

5. Deadlock Detection and Resolution Techniques:

- Implemented Wait-for graph algorithm in Python for deadlock detection.

- Developed resolution strategies, including transaction rollbacks and a retry mechanism after predefined intervals.

Team Collaboration:

The team of four divided the tasks to leverage individual strengths:

- Anurag focused on setting up the MySQL database schema and ensuring the integrity of the tables and relationships.
- Adruti was responsible for writing the Python scripts to handle transaction execution and simulate deadlocks.
- Moksha and Anurag developed the deadlock detection algorithm using the Wait-for graph approach.
- Adruti and Vaibhav implemented the deadlock resolution and retry mechanisms, and also set up the simulation environment in VS Code.

References

- Operating System Concepts, 9th Edition by Abraham Silberschatz, Peter B. Galvin, Greg Gagne Released December 2012
- Kate, V., Jaiswal, A. and Gehlot, A., 2016, March. A survey on distributed deadlock and distributed algorithms to detect and resolve deadlock. In 2016 Symposium on Colossal Data Analysis and Networking (CDAN) (pp. 1-6). IEEE.
- "Transaction Processing: Concepts and Techniques" by Jim Gray and Andreas Reuter
- Gupta, S., 2013. Deadlock detection techniques in distributed database system. International Journal of Computer Applications, 74(21).