Distributed Library System: Project Report

Deniz Aksoy (155896), Michał Redmer (156055)

1. Introduction

This report outlines the Distributed Library System, a fault-tolerant and scalable reservation platform for managing library resources. Developed for the "Big Data and Distributed Processing" course, it addresses the need for a robust system handling concurrent operations and large data loads in a distributed environment, leveraging modern web technologies and a distributed NoSQL backend.

2. Project Overview

The Distributed Library System enables users to manage library book reservations: viewing available books, making, updating, and canceling reservations (individually or in bulk), and tracking ownership. It also incorporates tools for comprehensive stress testing to evaluate performance under concurrent loads.

3. Requirements Fulfilled

The project successfully addresses key Distributed Database Application requirements, including the use of Apache Cassandra as a distributed database, a React and TypeScript frontend UI, and operation on a two-node Cassandra cluster. It supports core reservation management (making, updating, viewing, and canceling reservations, plus ownership tracking), ensures performance with an async, non-blocking Tornado backend for efficient error handling and minimal delays, and fully supports detailed stress test requirements with integrated tools.

4. System Architecture

The system follows a client-server architecture:

- **Frontend**: React and TypeScript, primarily using Tailwind CSS (with some SCSS). Handles user interaction and data fetching via Axios and React Query.
- **Backend**: Python with the Tornado framework. Stateless, handles business logic, API routing, and database communication. Its asynchronous nature ensures scalability.
- **Database**: A two-node **Apache Cassandra** cluster provides eventual consistency and high availability.
- **Cluster management**: Managed via Docker and `docker-compose.yml`.

5. Key Features

The system offers robust features, including secure and comprehensive reservation management, clear identification of reservation owners through user tracking, and critical stress testing capabilities (detailed in Section 6) for evaluating performance under demanding conditions. An asynchronous backend ensures responsiveness and high throughput, complemented by a visually appealing and functional user-friendly UI.

6. Stress Testing Implementation

The project heavily emphasizes stress testing to validate its distributed nature and resilience. Five specific tests are implemented:

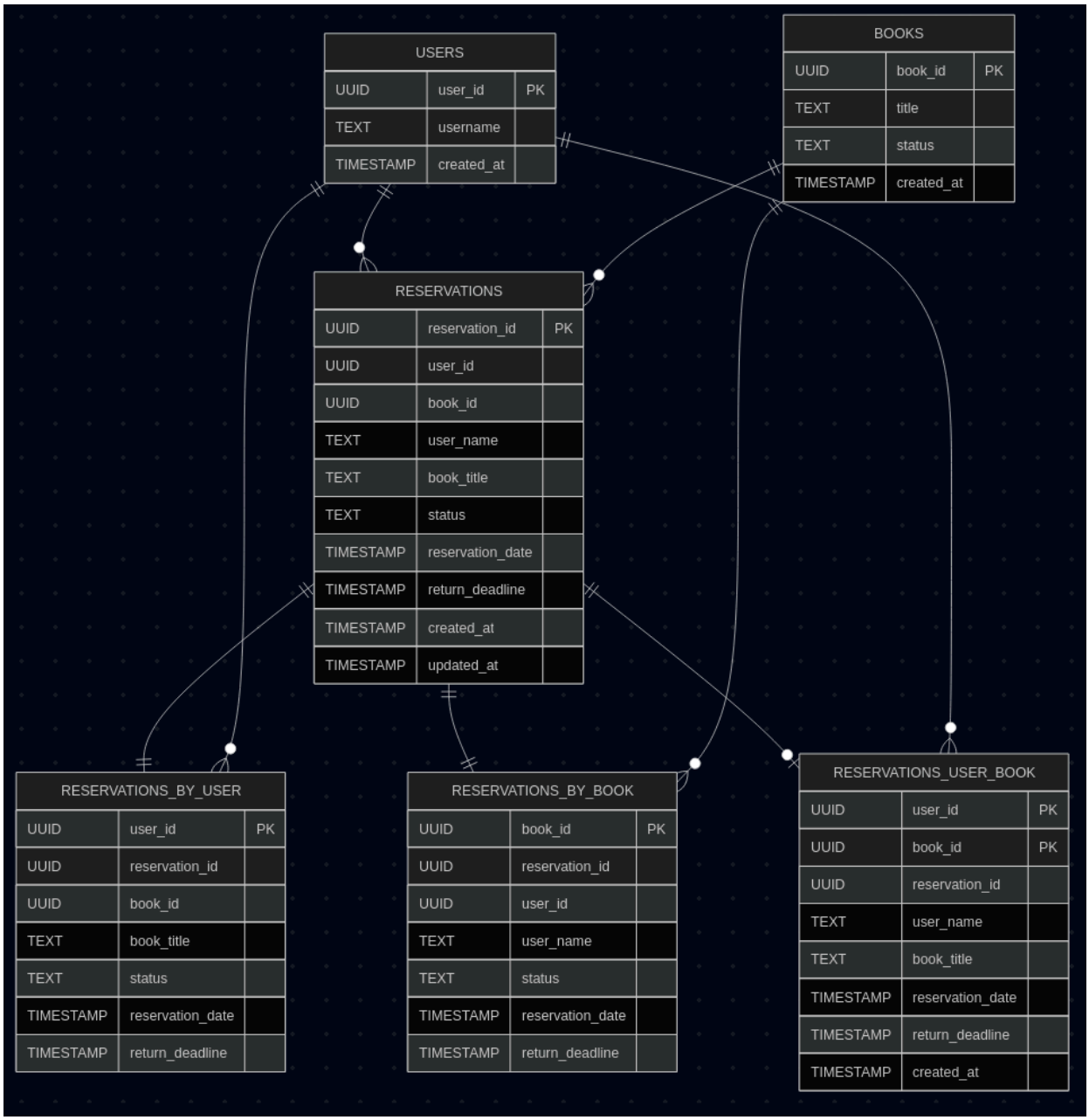
1. **Rapid repeated requests**: Tests handling bursts from a single client.
2. **Concurrent client races**: Assesses contention management with multiple clients.
3. **Immediate occupancy**: Ensures fair resource allocation during simultaneous reservation attempts by two clients.
4. **Constant cancellations and seat occupancy**: Tests consistency under continuous changes.
5. **Large group Cancellation**: Evaluates efficiency for bulk cancellations.

All implemented stress tests have passed with a **100% success rate over 1000+ requests**, confirming the system's reliability and performance under load.

7. Database Schema

6 Tables:
reservations - Main table with complete reservation data
reservations_by_user - complete reservation data optimized for querying by user
reservations_by_book - complete reservation data optimized for querying by book
reservations_user_book - Active reservations only, for fast lookups
books - Book catalog with availability status

users - User catalog



8. Challenges and Conclusions

Key challenges overcome during development include configuring Docker-Compose to ensure reliable backend-Cassandra connectivity, designing effective denormalized Cassandra schemas with multiple tables while maintaining consistency under concurrency for database structure, and creating a fully functional and robust API for all reservation operations. Additionally, building an intuitive and functional frontend UI with correct data fetching logic was a significant hurdle.

Finally, as a new area, developing comprehensive load simulations and ensuring 100% test pass rates for stress test creation and validation required significant effort and learning. The successful resolution of these challenges contributed significantly to the system's robustness, and its consistent performance under stress tests underscores its ability to meet the demands of a distributed environment.