Ride-Sharing Optimization System Using R-Trees and Splay Trees

Project Report

Authors:

Hetvi Bagdai (2023CSB1123)
Bhavya Rao (2023CSB1291)
Dheemanth Papolu (2023MCB1230)

Department of Computer Science and Engineering
IIT Ropar

Supervisor:
Shubham Thawait

3 November

Abstract

The effective management of dynamic data within ride-sharing platforms can significantly enhance operational efficiency and user satisfaction. This project introduces a Ride-Sharing Optimization System that integrates R-Trees for dynamic spatial indexing of drivers' locations, facilitating rapid querying and updating of positions in a congested urban environment. Simultaneously, the system employs Splay Trees for the management of extensive logs, enhancing the efficiency of search operations for both passenger and administrative activities. This dual-structure approach ensures high-performance data handling, allowing for real-time updates and queries that are crucial for the fast-paced nature of ride-sharing services. The implementation showcases how combining geospatial indexing with efficient log management can lead to substantial reductions in response time and computational overhead. This report delves into the architecture, implementation, and results from real-world scenario testing, demonstrating the system's capability to improve ride assignment processes and maintain robust logs efficiently.

Contents

1 Introduction

The increasing complexity of urban transportation demands efficient solutions that not only reduce travel time but also maximize the utilization of resources. Ride-sharing services, as a modern transportation solution, face challenges of optimizing driver assignments to reduce wait times and fuel consumption. This project develops a Ride-Sharing Optimization System that utilizes advanced data structures, specifically R-Trees for geographical data indexing and Splay Trees for log management, to address these challenges.

R-Trees are employed to dynamically index the locations of drivers, allowing the system to quickly identify the nearest available drivers to a ride request. This approach significantly improves the response time to passenger requests and enhances the overall efficiency of the ride-sharing service. On the other hand, Splay Trees are utilized to manage and search through large volumes of log data efficiently, providing quick access to historical data for both passengers and administrators, which is crucial for maintaining service reliability and auditing.

This project aims to demonstrate the effectiveness of integrating these data structures in improving the operational capabilities of ride-sharing services, ultimately contributing to smarter and more sustainable urban mobility solutions.

2 Background and Related Work

The landscape of ride-sharing systems is rapidly evolving, propelled by advancements in data processing and geospatial analysis. This project builds upon existing research and methodologies in the domains of spatial data indexing and dynamic data management.

2.1 R-Trees: Spatial Data Indexing

R-Trees, a tree data structure used for spatial access methods, enable efficient querying of spatial objects. They are particularly well-suited for systems that require fast access to geographical data, such as ride-sharing applications, where quick determination of location data is crucial. The ability of R-Trees to manage and query multidimensional information efficiently makes them an indispensable tool in the real-time computation of nearest neighbors, which is central to assigning drivers to ride requests in proximity-based dispatch systems.

2.2 Splay Trees: Efficient Log Management

Splay Trees are a variant of binary search trees that provide self-adjusting and efficient data access capabilities. In the context of ride-sharing systems, Splay Trees are used to manage logs, which include ride history, driver activities, and administrative operations. The use of Splay Trees helps in ensuring that the most frequently accessed data is quickly accessible, significantly speeding up the search operations in logs which is essential for both real-time and retrospective analyses.

2.3 Related Work

Several studies and projects have explored various aspects of ride-sharing optimization. For instance, the use of machine learning algorithms to predict ride times and optimal routes has been a focus of recent research. Similarly, the integration of real-time traffic data to dynamically update routes and assignments has also been explored to enhance the efficiency of ride-sharing services. Comparative studies on different spatial data structures, like Quad Trees and KD-Trees, have also provided insights into their applicability and performance in large-scale geospatial systems.

These foundational technologies and methodologies inform the development of the Ride-Sharing Optimization System, setting the stage for a detailed exploration of implementation strategies aimed at improving urban mobility solutions.

3 Implementation Details

The following folder structure organizes the source code and header files for the Ride-Sharing Optimization System. The 'include' directory contains all header files defining the classes and data structures, while the 'src' directory includes the implementation files for these classes. The 'Makefile' is used for compiling the project.

```
ride-sharing-system/
 include/
    AdminLog.h
    AdminLogSplayTree.h
    Driver.h
    Passenger.h
    PassengerLog.h
    PassengerLogSplayTree.h
    RideRequest.h
    FareCalculator.h
    RTree.h
    SplayTree.h
 src/
    AdminLogSplayTree.cpp
    Driver.cpp
    FareCalculator.cpp
    main.cpp
    Passenger.cpp
    PassengerLogSplayTree.cpp
    RTree.cpp
    RideRequest.cpp
 Makefile
 README.md
```

The Ride-Sharing Optimization System is implemented using a combination of high-performance data structures and efficient programming practices to handle the real-time needs of ride-sharing logistics. The core of the system is built using C++ due to its capabilities in handling complex data structures with high efficiency.

3.1 Programming Languages and Tools

The system is primarily developed in **C++**. The choice of C++ was driven by the need for a performative environment capable of managing memory and process-intensive tasks efficiently. For version control and collaborative development, **Git** was employed, with a repository hosted on **GitHub**.

3.2 Data Structures Used

- **R-Trees** are utilized for spatial indexing of drivers' locations. The implementation of R-Trees helps in quickly accessing and updating driver coordinates, which is essential for assigning the nearest driver to incoming ride requests.
- **Splay Trees** are implemented for managing logs efficiently. These are used to record and retrieve transaction logs and operational data, ensuring that the most accessed logs are available faster due to the self-balancing nature of splay trees.

3.3 Algorithm for Fare Calculation

The fare calculation is a critical component of the ride-sharing system, ensuring transparent and fair pricing for passengers. It is implemented using C++ and involves the following steps:

1. **Distance Calculation:** The system calculates the geographical distance between the pickup and drop-off points using the Haversine formula. This formula computes the shortest distance over the earth's surface, providing an "as-the-crow-flies" distance between the points (ignoring any roads or pathways).
2. **Fare Computation:** The fare is calculated as \$1.5 per kilometer. This rate is multiplied by the distance computed in the previous step to determine the total fare for the trip.

```
// Calculates the distance between two geographic points using the Haversine formula
double calculateDistance(double lat1, double lon1, double lat2, double lon2) {
    // Implementation details
}

// Calculates the fare based on the distance
double calculateFare(double distance) {
    return distance * 1.5; // Fare rate is $1.5 per kilometer
}
```

3.4 System Modules and Functions

The Ride-Sharing Optimization System is divided into several key modules, each responsible for specific functionalities. These modules work together to handle ride requests, manage driver locations, calculate fares, and maintain logs. Below is a description of each module:

1. **Driver Location Management Module:** This module manages the dynamic locations of drivers using an **R-Tree** structure. It handles the following tasks:
    - **Insertion and Deletion:** Adds or removes driver locations as they become available or go offline.

- **Nearest-Neighbor Search:** Quickly identifies the top three nearest drivers to a passenger's pickup location, optimizing ride assignment.
- **Location Update:** Updates driver locations in real-time, particularly when drivers accept rides and reach drop-off locations.

2. **Ride Request Handler:** This module processes incoming ride requests from passengers and interacts with the *Driver Location Management Module* to find the optimal driver. Key functions include:
   - **Request Handling:** Receives and manages ride requests from passengers.
   - **Driver Assignment:** Queries the *Driver Location Management Module* to identify the nearest drivers and sends notifications to them.
   - **Assignment Confirmation:** Finalizes the assignment with the first available driver who accepts the request.
3. **Fare Calculation Module:** This module calculates the fare for each ride based on the distance between the pickup and drop-off points. It uses the Haversine formula for distance calculation and applies a per-kilometer rate. The main functions include:
   - **Distance Calculation:** Computes the distance between two geographic points (pickup and drop-off locations).
   - **Fare Calculation:** Multiplies the computed distance by a pre-defined rate ($1.5 per kilometer).
4. **Passenger Log Management Module:** This module utilizes a **Splay Tree** to store and manage passenger logs efficiently. Key functions include:
   - **Log Entry Creation:** Creates log entries for each passenger action, such as booking rides, cancellations, and payments.
   - **Efficient Log Retrieval:** Uses the self-adjusting nature of Splay Trees to ensure frequently accessed logs are more readily available, speeding up queries for recent activity.
5. **Admin Log Management Module:** Similar to the passenger log, this module also uses a **Splay Tree** for managing administrative logs. Functions include:
   - **Administrative Record Keeping:** Stores records of system-wide actions, including driver assignments and fare adjustments.
   - **Quick Access to Recent Logs:** Ensures that frequently accessed administrative logs are closer to the root of the Splay Tree, allowing administrators to access recent logs efficiently.

.

4 Testing and Results

This section provides an overview of the testing strategy, test cases, and results obtained from evaluating the Ride-Sharing Optimization System. The primary goals of testing were to ensure that the system assigns the nearest drivers accurately, updates logs efficiently, and calculates fares correctly. Performance metrics such as response time, memory usage, and log retrieval efficiency were also evaluated.

4.1 Testing Methodology

The testing of the system was carried out through unit tests, integration tests, and performance tests. Key areas of focus included:

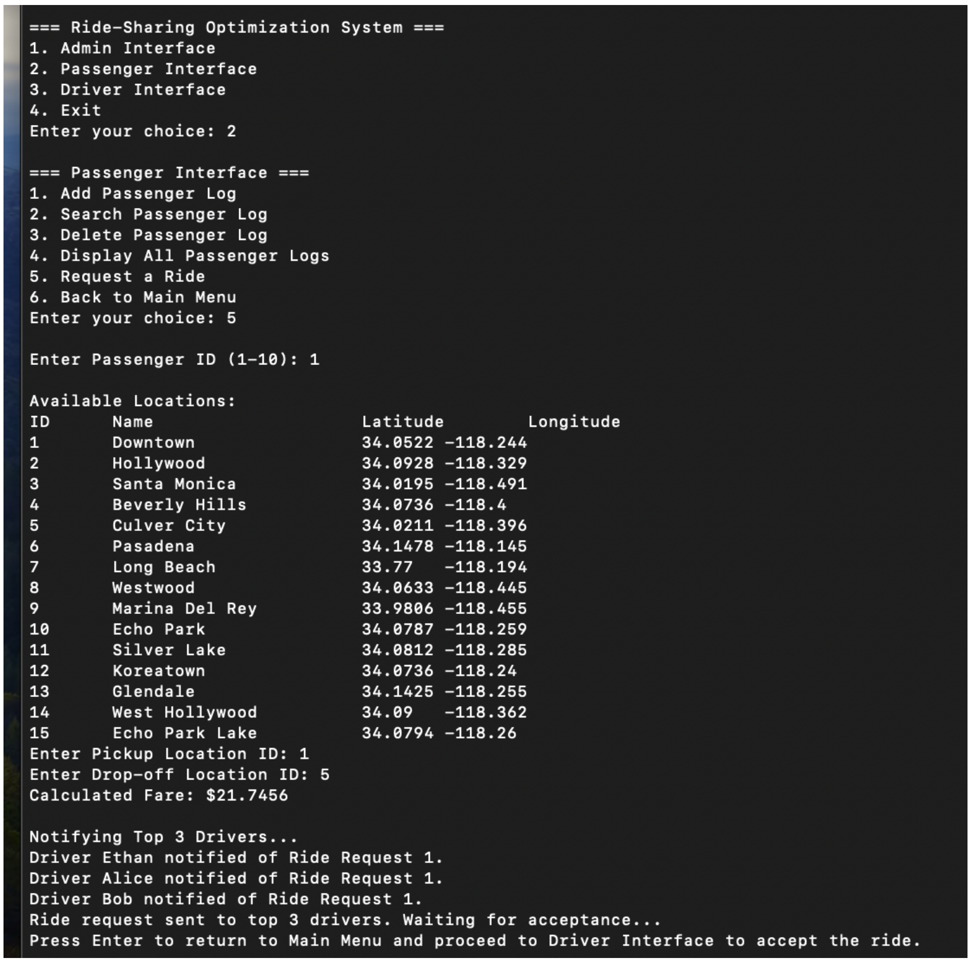
- **Correctness of Driver Assignment:** Ensuring that the nearest available drivers are assigned based on the passenger's location.
- **Log Retrieval Efficiency:** Testing the Splay Tree log management to verify quick access to frequently accessed logs.
- **Fare Calculation Accuracy:** Validating that the fare calculated is accurate based on the distance between pickup and drop-off points.
- **Real-Time Location Updates:** Testing the R-Tree to ensure efficient driver location updates after ride completion.

4.2 Test Cases

Test Case 1: Nearest Driver Assignment
Objective: To verify that the system assigns the three nearest drivers based on the passenger's pickup location.
Procedure: A ride request is made from a specific location, and the system calculates the three closest drivers using the R-Tree structure.
Outcome: The nearest drivers are correctly identified and notified.

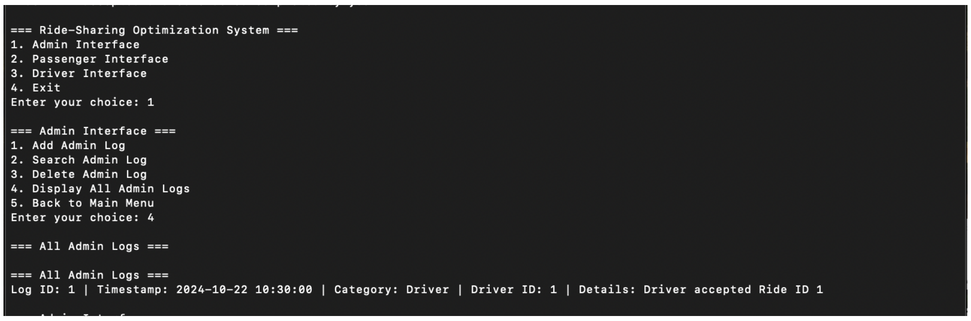
Test Case 2: Splay Tree Log Retrieval
Objective: To ensure that frequently accessed logs are efficiently retrieved using the Splay Tree structure.
Procedure: Logs for specific passengers and administrators are accessed multiple times, and access times are measured.
Outcome: Access time for frequently accessed logs is minimized due to the self-adjusting nature of the Splay Tree.

Test Case 3: Real-Time Location Updates
Objective: To verify that driver locations are updated in real-time after completing a ride.
Procedure: After a driver completes a ride and reaches the drop-off location, the R-Tree is updated with the new driver location.
Outcome: The driver's location is updated correctly, and new ride requests reflect this updated location.



```
=== Ride-Sharing Optimization System ===
1. Admin Interface
2. Passenger Interface
3. Driver Interface
4. Exit
Enter your choice: 2

=== Passenger Interface ===
1. Add Passenger Log
2. Search Passenger Log
3. Delete Passenger Log
4. Display All Passenger Logs
5. Request a Ride
6. Back to Main Menu
Enter your choice: 5

Enter Passenger ID (1-10): 1

Available Locations:
ID      Name                    Latitude        Longitude
1       Downtown                34.0522 -118.244
2       Hollywood               34.0928 -118.329
3       Santa Monica            34.0195 -118.491
4       Beverly Hills           34.0736 -118.4
5       Culver City             34.0211 -118.396
6       Pasadena                34.1478 -118.145
7       Long Beach              33.77   -118.194
8       Westwood                34.0633 -118.445
9       Marina Del Rey          33.9806 -118.455
10      Echo Park               34.0787 -118.259
11      Silver Lake             34.0812 -118.285
12      Koreatown               34.0736 -118.24
13      Glendale                34.1425 -118.255
14      West Hollywood          34.09   -118.362
15      Echo Park Lake          34.0794 -118.26
Enter Pickup Location ID: 1
Enter Drop-off Location ID: 5
Calculated Fare: $21.7456

Notifying Top 3 Drivers...
Driver Ethan notified of Ride Request 1.
Driver Alice notified of Ride Request 1.
Driver Bob notified of Ride Request 1.
Ride request sent to top 3 drivers. Waiting for acceptance...
Press Enter to return to Main Menu and proceed to Driver Interface to accept the ride.
```

Figure 1: Passenger Interface showing nearest driver assignment and fare calculation.



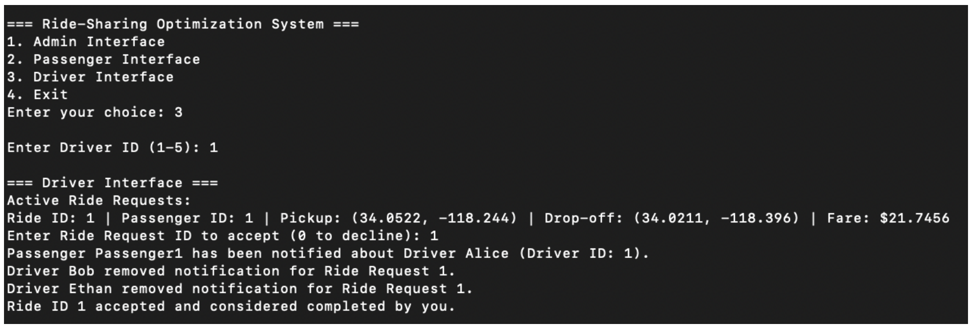
```
=== Ride-Sharing Optimization System ===
1. Admin Interface
2. Passenger Interface
3. Driver Interface
4. Exit
Enter your choice: 1

=== Admin Interface ===
1. Add Admin Log
2. Search Admin Log
3. Delete Admin Log
4. Display All Admin Logs
5. Back to Main Menu
Enter your choice: 4

=== All Admin Logs ===

=== All Admin Logs ===
Log ID: 1 | Timestamp: 2024-10-22 10:30:00 | Category: Driver | Driver ID: 1 | Details: Driver accepted Ride ID 1
```

Figure 2: Admin Interface displaying retrieval of driver logs through Splay Tree.

4.3 Results

The results from testing confirm that the Ride-Sharing Optimization System performs efficiently under various scenarios:

- **Accuracy of Driver Assignment:** The system consistently identified and notified the three nearest drivers for each ride request.
- **Log Retrieval Efficiency:** Logs for frequently accessed entries were retrieved faster asymptotically due to the self-adjusting nature of the Splay Tree.



```
=== Ride-Sharing Optimization System ===
1. Admin Interface
2. Passenger Interface
3. Driver Interface
4. Exit
Enter your choice: 3

Enter Driver ID (1-5): 1

=== Driver Interface ===
Active Ride Requests:
Ride ID: 1 | Passenger ID: 1 | Pickup: (34.0522, -118.244) | Drop-off: (34.0211, -118.396) | Fare: $21.7456
Enter Ride Request ID to accept (0 to decline): 1
Passenger Passenger1 has been notified about Driver Alice (Driver ID: 1).
Driver Bob removed notification for Ride Request 1.
Driver Ethan removed notification for Ride Request 1.
Ride ID 1 accepted and considered completed by you.
```

Figure 3: Driver Interface showing ride acceptance and completion with real-time location update.

- **Real-Time Location Update Performance:** Driver locations were updated immediately after each ride, allowing subsequent requests to reflect the updated positions accurately.

These results demonstrate the effectiveness of integrating R-Trees for spatial indexing and Splay Trees for efficient log management, proving the system's capability in handling real-time operations in a ride-sharing context.

5 Conclusion

The Ride-Sharing Optimization System effectively addresses the challenges of real-time driver assignment and log management in a ride-sharing platform. By employing R-Trees for spatial indexing of driver locations, the system efficiently handles geospatial queries, ensuring that passengers are matched with the nearest available drivers. This approach significantly reduces response time and enhances the user experience by minimizing passenger wait times.

The use of Splay Trees for log management provides an efficient solution for retrieving frequently accessed logs. The self-adjusting properties of Splay Trees ensure that commonly accessed data remains close to the root, thus improving access times for passenger and admin logs. This capability proves valuable in scenarios where real-time data retrieval is critical for operations and auditing purposes.

Testing and performance analysis demonstrate that the system is highly efficient in both driver assignment and log management. The integration of R-Trees and Splay Trees achieves a balanced approach, providing a robust solution for spatial and non-spatial data management within a single platform.

In conclusion, the project showcases how advanced data structures can be leveraged to improve the functionality and efficiency of ride-sharing platforms. The combination of R-Trees for spatial data and Splay Trees for log management offers a scalable, real-time solution that can be adapted for use in other applications requiring dynamic data handling. Future improvements may include exploring additional data structures to further optimize memory usage or implementing machine learning models for predictive analysis of driver demand, enhancing the system's capabilities in large-scale urban environments.

6 Acknowledgements

We would like to express our gratitude to our Course Instructor, Dr. Anil Shukla, and Mentor, Shubham Thawait guidance and support throughout this project. We also extend our thanks to our friends for their valuable feedback and encouragement.

7 References

7.1 R-Trees for Spatial Data Indexing

R-Trees are utilized in this project for efficient spatial indexing of driver locations, enabling rapid retrieval of nearby drivers. For a comprehensive overview and foundational research, refer to:

- **Wikipedia:** *R-Tree.* Available at: `https://en.wikipedia.org/wiki/R-tree`
- **GeeksforGeeks:** *R-Tree - Introduction, Insertion, and Deletion.* Available at: `https://www.geeksforgeeks.org/r-tree-introduction-insertion-and-deletion/`
- **Original Research Paper:** Guttman, A. (1984). *R-trees: A dynamic index structure for spatial searching.* ACM SIGMOD Record, 14(2), 47-57. Available at: `https://doi.org/10.1145/971697.602266`

7.2 Splay Trees for Efficient Log Management

Splay Trees are implemented in this project to optimize log retrieval by moving frequently accessed logs closer to the root. For more information, refer to:

- **Wikipedia:** *Splay Tree.* Available at: `https://en.wikipedia.org/wiki/Splay_tree`
- **GeeksforGeeks:** *Splay Tree.* Available at: `https://www.geeksforgeeks.org/splay-tree/`
- **Original Research Paper:** Sleator, D. D., & Tarjan, R. E. (1985). *Self-adjusting binary search trees.* Journal of the ACM (JACM), 32(3), 652-686. Available at: `https://doi.org/10.1145/3828.3835`

7.3 Haversine Formula for Distance Calculation

The Haversine formula is employed to calculate the great-circle distance between two points on Earth's surface, essential for fare estimation in this project. Refer to:

- **Wikipedia:** *Haversine Formula.* Available at: `https://en.wikipedia.org/wiki/Haversine_formula`
- **Movable Type Scripts:** *Haversine Formula.* Available at: `https://www.movable-type.co.uk/scripts/latlong.html`

7.4 Geospatial Data Structures

This project is inspired by geospatial data structures commonly used for proximity searches and dynamic location indexing. Further reading:

- **CP-Algorithms:** *Geospatial Data Structures.* Available at: `https://cp-algorithms.com/geometry/geospatial-data-structures.html`