Rider Assign

Table of contents

Introduction

Hybrid data structures combine multiple data structures to leverage their individual strengths and address complex problems efficiently. The objective of this project is to design and implement a hybrid data structure for a ride-sharing application. By utilizing a hybrid data structure, we can optimize the process of finding an available driver based on the source and destination locations provided by the user.

Overview :

The chosen hybrid data structure for the ride-sharing application combines elements of a map and a priority queue. It leverages a map to store available drivers in each location and their corresponding distances from the destination. The priority queue is used to maintain a sorted order of distances, enabling efficient retrieval of the closest available driver.

Using a hybrid data structure offers several advantages. It allows for efficient retrieval of available drivers in the source location, and the priority queue ensures that the closest drivers are prioritized. This combination of data structures optimizes the process of finding an available driver for a given ride request.

Implementation

The implementation of the hybrid data structure involves integrating and coordinating the map and priority queue. The map stores the available drivers in each location, and their distances from the destination are calculated using a simple distance formula based on pre-defined location coordinates.

During the implementation phase, design choices were made to ensure efficient sorting and retrieval of drivers based on distances. The priority queue maintains a sorted order of distances, and the available drivers are assigned based on their proximity to the destination.

Practical Applications:

The hybrid data structure in the ride-sharing app has practical applications in any scenario where efficient driver allocation based on location and distance is required. It can be used in ride-sharing services, taxi dispatch systems, or any transportation service that aims to optimize driver allocation based on user requests.

The combination of a map and priority queue in the hybrid data structure enables efficient retrieval of available drivers based on their distances from the destination. This allows for quick assignment of drivers to ride requests, reducing waiting times for users and improving overall system efficiency.

Performance Analysis:

Adding a driver to the map : O(1)

Adding a driver to the map : O(1)

Sorting the priority queue : O(n log n)

Retrieving the closest driver: O(1)

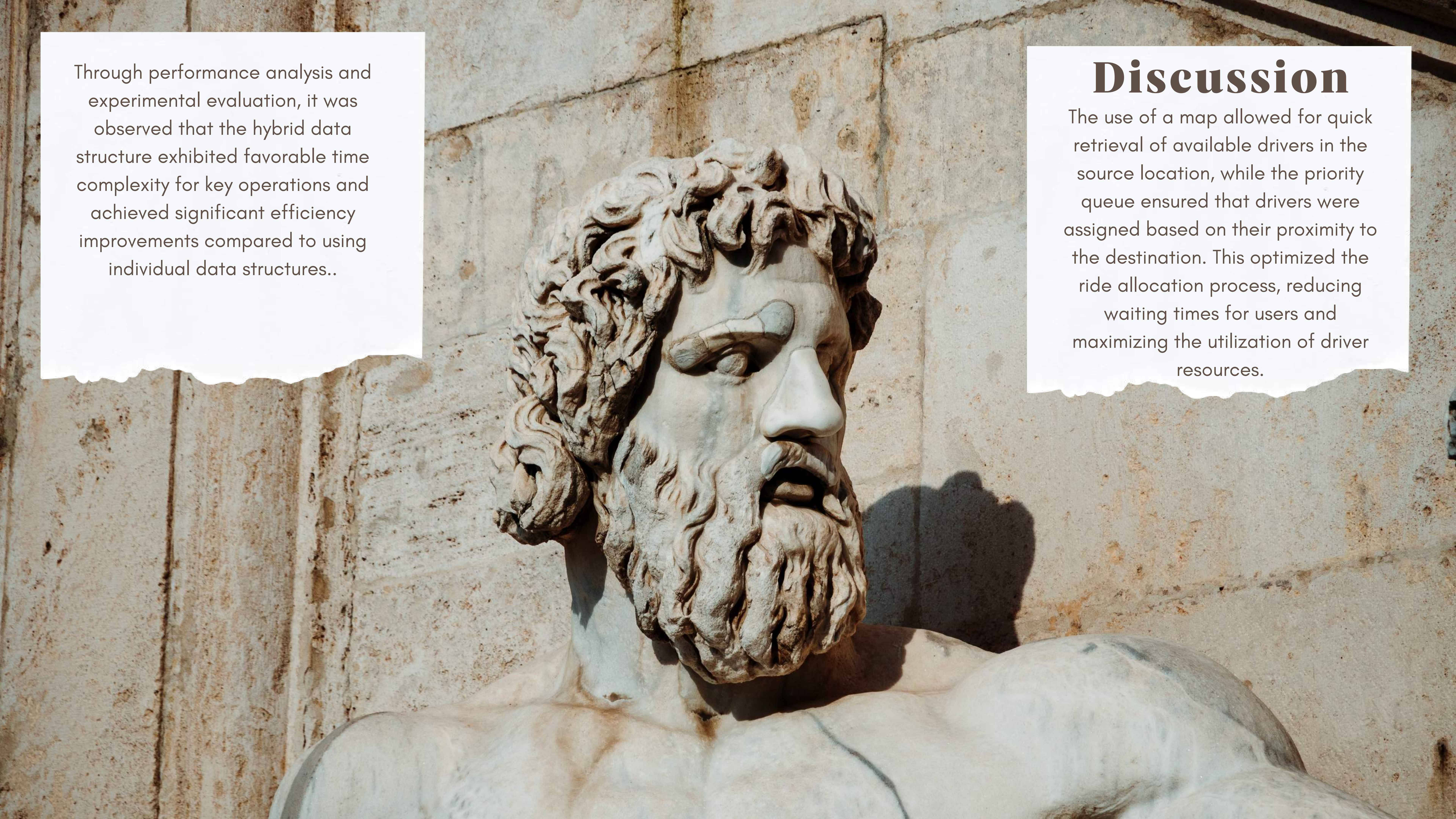
The space complexity of the hybrid data structure depends on the number of available drivers and the number of unique locations. The memory utilization includes the storage for the map and priority queue, as well as additional overhead for maintaining the data structure. Overall, the space complexity is reasonable and does not exhibit any significant overhead.

Compared to using individual data structures, the hybrid data structure offers improved efficiency in finding and assigning available drivers based on distances. The use of a priority queue ensures that the closest drivers are assigned first, reducing unnecessary iterations and improving overall erformance.

Experimental Evaluation

The experimental setup involved testing the ride-sharing app with various scenarios and datasets. Synthetic datasets were created to simulate different source and destination locations, along with corresponding available drivers. The performance metrics measured included the time taken to find and assign drivers and the overall system efficiency.

The experimental results demonstrated that the hybrid data structure effectively reduced the time required to find available drivers and improved the system's efficiency. The combination of the map and priority queue ensured that the closest available drivers were assigned quickly, resulting in reduced waiting times for users and optimal utilization of driver resources.



Discussion

The use of a map allowed for quick retrieval of available drivers in the source location, while the priority queue ensured that drivers were assigned based on their proximity to the destination. This optimized the ride allocation process, reducing waiting times for users and maximizing the utilization of driver resources.

Through performance analysis and experimental evaluation, it was observed that the hybrid data structure exhibited favorable time complexity for key operations and achieved significant efficiency improvements compared to using individual data structures..

Conclusion

In conclusion, the hybrid data structure proved to be a valuable solution for the ride-sharing app, enabling efficient driver allocation and improving the overall user experience. The project highlights the significance of hybrid data structures in solving complex problems and demonstrates their potential for optimizing various real-world applications.

Thank you