Case Study

Smart Cab Allocation System for Efficient Trip Planning

Submitted by Ujjwal Hendwe (IIT Indore)

A. Admin's Cab Allocation Optimization:

1. Data Collection and Real-time Updates: -

Acquire Cab Location Data:
Obtain real-time GPS data from all accessible cabs.
Gather Trip Requests:
Acquire trip requests together with their respective starting locations.

2. Cab Allocation: -

Compute Distance Matrix:
Utilize existing technologies (such as the Google Maps API) to determine the distances between each cab and the starting locations of the trips.
Locate Closest Taxi:
Determine the closest taxi available for each trip request by calculating the distances.

3. Allocation Decision: -

Assign Trip:
Allocate the closest taxi to the request for a trip and designate the taxi as occupied.
Update Cab Status:
Exclude the allocated cab from the group of accessible cabs.

4. Real-time Updates and Monitoring: -

Ongoing Updates:
Regularly refresh the taxi location data to accurately represent current changes in real-time.
Manage Incoming Trip Requests:
Continuously handle incoming trip requests and assign taxis depending on their proximity.

5. Optimisation and Efficiency Improvement: -

Optimisation Techniques:

Employ algorithms to distribute the workload evenly or redirect dynamically to enhance cab allocation and reduce total travel distance.

Monitor and Analyze:

Continuously observe trip distance, duration, and other pertinent metrics to assess the algorithm's efficacy.

6. Testing and Evaluation: -

Simulation Testing:

Employ historical or synthetic data to simulate diverse scenarios and assess the algorithm's performance under varied circumstances.

Empirical Validation:

Implement the algorithm in a controlled setting and assess its efficacy by comparing the actual travel lengths pre and post implementation.

A pseudo code for the above explanation can be written as :

```
function allocateCab(tripRequest):
    cabDistances = {}
    for each cab in availableCabs:
        distance = calculateDistance(cab.location, tripRequest
            .startLocation)
        cabDistances[cab] = distance

    nearestCab = findNearestCab(cabDistances)

    if nearestCab is not null:
        allocate(nearestCab, tripRequest)
        updateCabStatus(nearestCab)
    else:
        // No available cabs at the moment, handle accordingly

function calculateDistance(start, end):
    // Use a distance calculation method (e.g., Haversine formula)
    // to find the distance between two coordinates
    // Return the distance

function findNearestCab(cabDistances):
    // Find and return the cab with the minimum distance
    return cab with minimum distance in cabDistances

function updateCabStatus(cab):
    // Mark the assigned cab as occupied
    cab.status = "occupied"
```

Considerations: -

Scalability: Ensure the algorithm is capable of efficiently processing a substantial volume of simultaneous trip requests and taxi changes.

Resilience: Take into consideration various situations such as traffic congestion, road blockages, or unexpected fluctuations in the availability of taxis.

User Experience: Optimize client happiness by minimizing both wait times and travel lengths, while ensuring fairness in the distribution of cabs.

To optimize taxi allocation, minimize travel distance, and enhance overall trip efficiency, it is essential to follow these steps and constantly refine the algorithm using real-world data and feedback. Testing and iteration are essential to optimize the algorithm for improved performance over time.

B. Employee's Cab Search Optimization:

To optimize the user experience for workers seeking taxis and recommending nearby taxis that are currently occupied, please consider the following approach:

1. Real-time Data Integration:
- Collect Cab Data: Perpetually acquire up-to-the-minute GPS data from all taxis, including their present condition (occupied or available).
- Employee Location: Retrieve the present whereabouts of the employee in need of a taxi.

2. Nearby Cab Suggestions:
- Locate Nearby Cabs: Determine the proximity between the employee's present position and the locations of all cabs that are currently occupied.
- Proximity Filtering: Show taxis that are located close to the employee's current position, within a specified radius or distance limit.
- Rank Suggestions: Prioritize the occupied taxis in the vicinity according to their proximity to the employee's current location.

3. Presentation and User Interface:
- Visual Representation: Display the suggestions in a readily accessible and user-friendly interface, such as a map or a list arranged by proximity.
- Real-time Updates: Consistently refresh the suggestions as the employee's location changes or as new taxis become occupied in close proximity.

4. Evaluation of Effectiveness:
- input Collection: Obtain input from workers evaluating the proposed taxi system.
- Metrics Evaluation: Assess the duration it takes for employees to locate and select a taxi using the recommended strategy in comparison to the prior approach.
- Accuracy Assessment: Assess the precision of the recommendations by comparing the proposed nearby taxis with the actual available choices and their distances.

The Pseudo code can be written as:

```
function suggestNearbyCabs(employeeLocation):
    nearbyOccupiedCabs = {}

    for each cab in occupiedCabs:
        distance = calculateDistance(cab.location, employeeLocation)
        if distance <= MAX_RADIUS: // Define a maximum radius for nearby
            suggestions
            nearbyOccupiedCabs[cab] = distance

    sortedCabsByDistance = sortCabsByDistance(nearbyOccupiedCabs)

    return sortedCabsByDistance // Return a list of nearby occupied cabs
        sorted by distance

function calculateDistance(start, end):
    // Use a distance calculation method (e.g., Haversine formula)
    // to find the distance between two coordinates
    // Return the distance

function sortCabsByDistance(nearbyOccupiedCabs):
    // Sort the nearby occupied cabs based on their distance from the
        employee location
    // Return a sorted list of cabs
```

Considerations:

- Privacy and Security: It is important to ensure that the system complies with privacy standards and does not jeopardize the security of employee or cab driver data.

- Scalability: Develop the system to efficiently manage a substantial volume of simultaneous users and taxi changes.

- User Experience Testing: Perform iterative user testing and collect input to enhance and optimize the system's usability.

Through the implementation of these procedures and the ongoing collection of feedback and data, you may assess the efficiency of the system in delivering prompt and pertinent taxi recommendations to employees as they search for taxis in close proximity to their current position. Modifications and enhancements can be implemented depending on the gathered data and user input to optimize the overall user experience.

C.Real-Time Location Data Integration:

Incorporating live location data for taxis and route starting points necessitates a strong system capable of managing ongoing changes, guaranteeing precision, and tackling possible obstacles. Below is a concise summary of the necessary procedures to do this:

1. Real-Time Cab Location Tracking System: GPS Tracking: Develop a system that consistently monitors the GPS coordinates of all taxis in real time.
Data Transmission: Implement a robust mechanism (such as APIs or web sockets) to enable taxis to regularly send their position data to the central server.
Error Handling: Incorporate provisions for situations such as weak GPS signal or errors in data transfer, and establish error handling methods to guarantee the integrity of the data.

2. Updates on Trip Start Location: Continuously receive trip requests from users, including their specified start locations.
Location Validation: Verify and cleanse trip start locations to guarantee precision and uniformity.

3. Incorporation with Cab Allocation Algorithm:
Real-time cab position data and trip start locations should be included into the cab allocation algorithm for live data integration.
Continuously revise the distance matrix between taxis and trip starting points by including real-time location changes.
Dynamic Allocation: Revise the allocation method to incorporate the latest location data in order to propose neighboring taxis and assign trips.

4. Addressing Data Latency and Inaccuracies: Optimize data processing by implementing effective approaches to reduce the delay in updating and processing real-time location data.
Methods for rectifying errors: Utilize prediction algorithms or conduct historical data analysis to rectify any errors or anomalies in the location data.
Utilize filters or thresholds to exclude possibly unreliable or obsolete location data.

5. System Reliability and Redundancy: Secondary Systems: Deploy backup servers or redundancy protocols to guarantee uninterrupted operation in the event of server malfunctions or data loss.

Implement monitoring systems to promptly identify deviations, discrepancies, or malfunctions in real-time and provide warnings for rapid response.

6. Ongoing Enhancement and Evaluation:
Performance Evaluation: Consistently assess the system's performance through the monitoring of data correctness, system responsiveness, and user input.
Iterative Updates: Utilize acquired insights from evaluations to implement incremental enhancements, fine-tune algorithms, and optimize data processing in order to enhance system dependability and accuracy.

Pseudo Code for the same can be written as:

```
function updateCabLocation(cabID, newLocation):
    // Function to update the location of a specific cab in real time
    cab = getCabByID(cabID)
    cab.location = newLocation
    // Update the cab's location in the system

function receiveTripRequest(employeeLocation):
    // Function to receive a trip request with the employee's location
    tripRequest = createTripRequest(employeeLocation)
    allocateCab(tripRequest) // Proceed with allocating a cab based on
        the request

function allocateCab(tripRequest):
    cabDistances = {}

    for each cab in availableCabs:
        distance = calculateDistance(cab.location, tripRequest
            .startLocation)
        cabDistances[cab] = distance

    nearestCab = findNearestCab(cabDistances)

    if nearestCab is not null:
        allocate(nearestCab, tripRequest)
        updateCabStatus(nearestCab)
    else:
        // Handle case where no available cabs are nearby
```

```
function calculateDistance(start, end):
    // Function to calculate distance between two coordinates
    // Use appropriate distance calculation method (e.g., Haversine
        formula)

function updateRealTimeData():
    // Continuously updates the system with real-time location data
    while true:
        for each cab in allCabs:
            newLocation = getNewLocationFromGPS(cab)
            updateCabLocation(cab.ID, newLocation)
        // Repeat at intervals to update cab locations

        // Additional process to handle trip requests in real time
        for each employee in employees:
            employeeLocation = getCurrentLocation(employee)
            receiveTripRequest(employeeLocation)
        // Check for new trip requests and process them

        sleep(interval) // Wait for the specified interval before
            updating again
```

To achieve the smooth integration of real-time location data for taxis and trip start locations into the cab allocation system, it is crucial to follow these procedures, continually improve the system using up-to-date data, and swiftly tackle any difficulties that arise. This will enhance the accuracy and reliability of the system.