DECENTRALIZED SYSTEM TO COMPUTE THE SAFEST PATH

Abstract

We have designed and implemented a backend application to show people how to avoid dangerous spots on city streets while walking from one place to another. Here we are providing the paths that offer trade-offs between safety and distance. We have developed an algorithm that would give a person walking through a city options for getting from one place to another — the shortest path, the safest path that balanced between both factors.

Anushka Patil, Snehal Vyawahare, Keyur Mehta

Table of Contents

1. Introduction

Now-a-days we come across various social harm events like crime, traffic crashes, medical emergencies, drug use etc. Police and various other department are constantly involved in mitigating social harm issues using various techniques. So, it very important to come up with new algorithms, software systems and tools to overcome these issues.

In this project, we have designed a backend model which will take into consideration the crime data provided by the Indianapolis Metropolitan Police Department [1], which is been simulated using Hawkes process. This data is used to calculate the crime cost of any route using the Google API [2] maps to find all the incidents within 1 mile of the GPS coordinates. Using this cost later shortest and safest path is computed.

2. Problem Statement

Current navigation tools depend solely on real-time traffic conditions for computing the fastest route between point A and point B. Crowd-sourced traffic data is used to give the users the quickest route. [3] In some cases, the route is modified in the middle of the journey to reduce the travel time, based on newly formed congestions somewhere ahead in the route. However, the time of journey as the only factor of consideration is not enough metric in today's times.

Parameters like crime, climatic conditions and time of travel must all be considered for computing the best possible route. Drivers can be given various options of routes with information about what to expect in terms of both time of travel, climate and route safety.

3. Implemented Solution

Here we have implemented a solution where we have a decentralized system which computes the safest route for pedestrians and drivers based on previous data. Routes are compared based on crime rate and the time of journey.

The routes data will be fetched from Google Maps API about the possible routes between the source and destination and time of each segment of the route will be considered as set of inputs. The data from machine learning algorithm when applied to these live inputs will give us expected crime cost for the entire route, which will be used for further calculating the crime factor of the route. The crime factor gives the user a metric to compare the routes and displays the duration of travel alongside the summary of the route to the user. This system is reliable and fault tolerant.

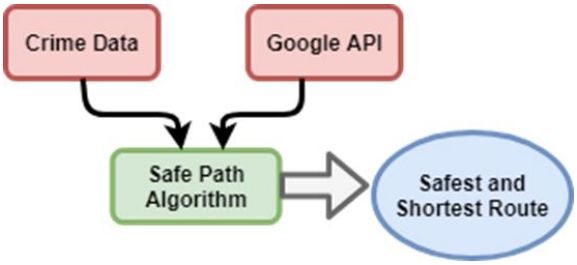


Figure 1 Implemented Solution

4. Design Details

Our design implements the concept of *separation of concerns* design principle. Each component in our system addresses a separate functionality which simplifies the development and maintenance of our system. For communication between the different components, is done using *Java RMI* [3] communication paradigm.

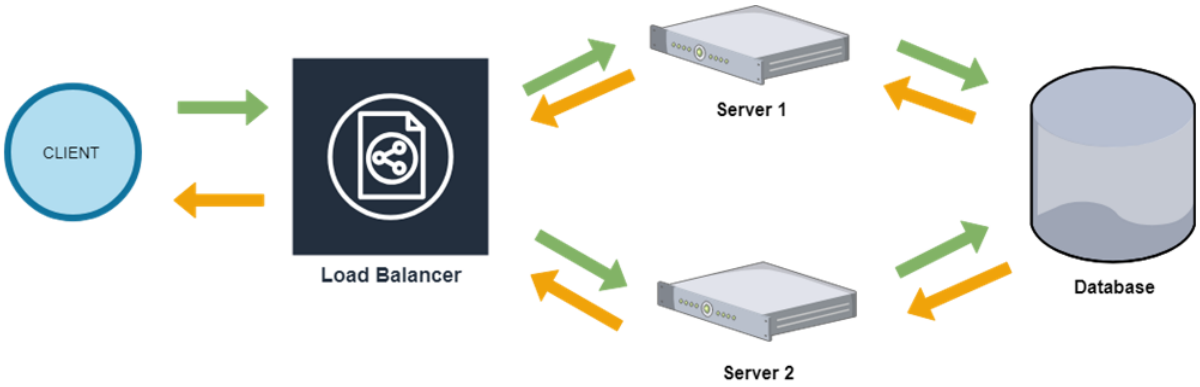


Figure 2 Architectural Design model

The system comprises of following components:

1. Client

The client provides an interface which allows the user to request services of the server. In our case, it requests the server for a safest short route to commute from source to destination.

2. Load Balancer

The load balancer resides in front of our servers and delegates client requests to either of the servers. The load balancer makes the best use of server capacity by ensuring that no one server is overworked. Our load balancing algorithm works on the least recently used mechanism i.e. the request is distributed based on which server was least recently used. Thus, the load balancer distributes client requests efficiently across both the servers. Additionally, it is capable of handling the server-side failures. If one server goes down or crashes, it redirects the request to another server.

3. Application Server

The application server is a server program which provides the application logic for our system. The server receives the client request via the load balancer. On reception of this request, the application server makes a request call to the Google Maps API [2] to obtain a set of routes for the requested source-destination. Further, it executes a database call to obtain incidents corresponding to the routes. It runs this data through our safe route algorithm and the safest short route is obtained. It stores this route in the client's session object and returns it to the load balancer.

4. Database Server

The database server provides the incident data [1] requested by the application server. Additionally, it is responsible for performing all the query processes. The database server maintains and retrieves the crime data.

5. Backup Server

The backup server is a replica of the application server. It can perform all the operations as the application server. In cases where the application server crashes, the load balancer redirects the client request to the backup server. The backup server performs all the necessary operations and returns the safest short route to the load balancer. On the other hand, the client is unaware of the crash and redirection to the backup server. Thus, providing failure transparency.

5. Implementation Details

The overall idea of our project is to avoid dangerous spots on city streets while moving from one place to another. The backend model relies both on crime data when it's available, and crowdsourced comments to reveal potential trouble spots to users.

Our implementation consists of two phases:

Phase 1: Extraction of crime data

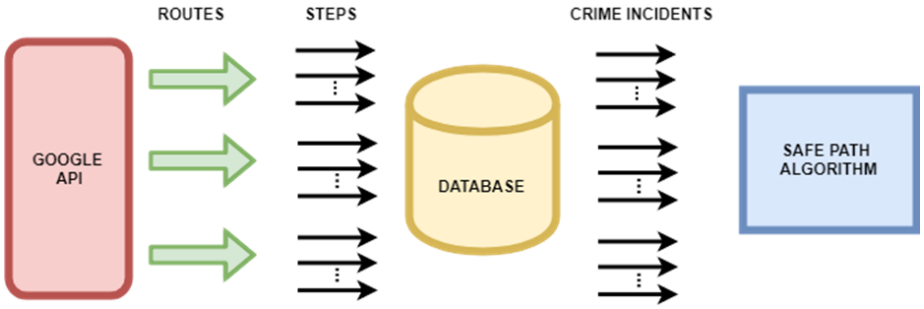


Figure 3 Extraction of crime data

The application server performs a request call to the Google API. [2] The Google API responds a JSON reply which contains three routes for the corresponding source-destination. Each route contains steps which are basically waypoints for that route, it gives the directions of the route. The server extracts all the latitude and longitudes for each step from each of the routes. Further, a query is executed to obtain crime incident data at the radius of 1 mile for each step. This data is then forwarded to the algorithm for safe path computation.

Phase 2: Safe Path Algorithm

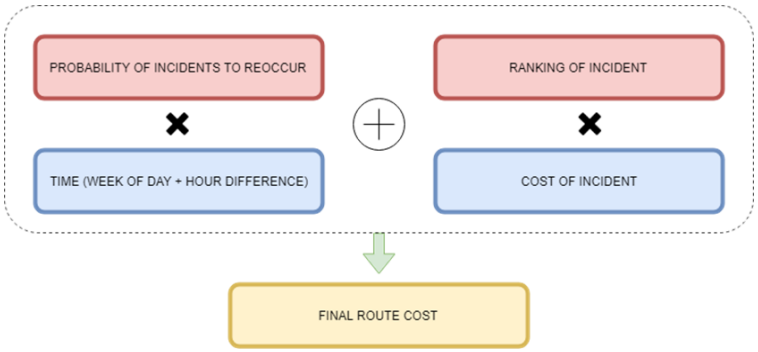


Figure 4 Safe path Algorithm

The data obtained from the database server contains incidents and time. Each incident is ranked priority wise and has a cost value associated with it.

Step 1: Normalization of values
The ranking of each incident and cost associated with it is normalized. Min-max normalization strategy is used to normalize values between 0 and 1.

Step 2: Time computation
Day of the week and the hour difference is calculated. In the case of the day of the week, if both days are same then it is put on higher priority else if both days are either working days or weekdays it is put on medium priority else on low priority. Further in the case of hours, if the hour difference is less than 5 hours it is put on higher priority else if greater than 5 hours and less than 12 hours it is put on medium priority else on low priority. Likewise, the time is computed for each incident.

Step 3: Probability of incidents to reoccur
The probability of incidents to reoccur is computed using the probability formula for an event to occur, which is the total number of times the events occurred by the total number of events. Using this formula, we have calculated the probability of incidents to reoccur on a given route.

Step 4: Final cost computation
The final cost is the summation of the product of the probability of incidents to reoccur and time along with the product of the ranking of incident and cost associated with it.

Once the final cost for each route is computed. The route array is sorted with respect to the distance of each route i.e. route [0] is the shortest route.
The selection of the Safest Shortest Route is based on the following criteria:

- If the differences between the cost of the safest route and the shortest route are less than 15% than the shortest route is recommended.
- If the difference is greater than 15%, the safest route is recommended.

The route information is returned to the client in JSON format similar to the Google API response.

6. Results Discussion

1. Safest Path and Shortest Path

In ideal case, the safe Path Algorithm will return the path having the lowest crime cost. In above result, though the shortest path is of 6.0 mi the, it has returned the safest path having distance of 6.1 mi.

```
========================================
Welcome to the Navigation Application
========================================
Enter the Source:
walmart, Indianapolis
Enter the Destination:
IUPUI

========================================
The distance of the safest and shortest route is: 6.1 mi
========================================
```

```
mehtake@thunder:~/DC/SafePathServer/SafePath
The response code is : 200
Get all steps of all routes...
Total routes retrieved: 3
Getting crime incidents on all routes...

========================================
The distance of route 0 is: 8.3
The cost of route 0 is: 210233.750708019
The distance of route 1 is: 6.0
The cost of route 1 is: 135832.45450936558
The distance of route 2 is: 6.1
The cost of route 2 is: 107590.81754551746
========================================
Sorted Routes
The distance of route 0 is: 6.0
The cost of route 0 is: 135832.45450936558
The distance of route 1 is: 6.1
The cost of route 1 is: 107590.81754551746
The distance of route 2 is: 8.3
The cost of route 2 is: 210233.750708019

========================================
Return route: 2
```

2. Safest Path (Difference between the safest and shortest path is less than 15%)

The above example shows that the algorithm will return the shortest route instead of safest route when the crime cost difference between them is less than 15%.

```
========================================
Welcome to the Navigation Application
========================================
Enter the Source:
Kroger Indianapolis
Enter the Destination:
target Indianapolis

========================================
The distance of the safest and shortest route is: 7.7 mi
========================================
```

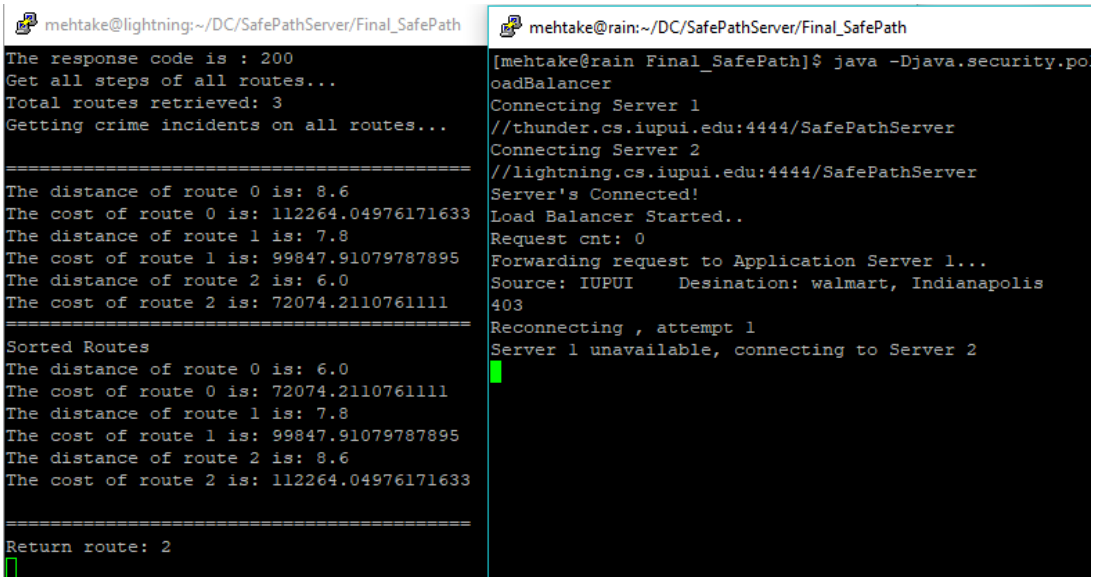
```
mehtake@thunder:~/DC/SafePathServer/SafePath
The response code is : 200
Get all steps of all routes...
Total routes retrieved: 3
Getting crime incidents on all routes...

========================================
The distance of route 0 is: 9.1
The cost of route 0 is: 280356.3448760187
The distance of route 1 is: 7.7
The cost of route 1 is: 312214.42838988546
The distance of route 2 is: 8.5
The cost of route 2 is: 406956.013266359
========================================
Sorted Routes
The distance of route 0 is: 7.7
The cost of route 0 is: 312214.42838988546
The distance of route 1 is: 8.5
The cost of route 1 is: 406956.013266359
The distance of route 2 is: 9.1
The cost of route 2 is: 280356.3448760187

========================================
Return route: 1
```

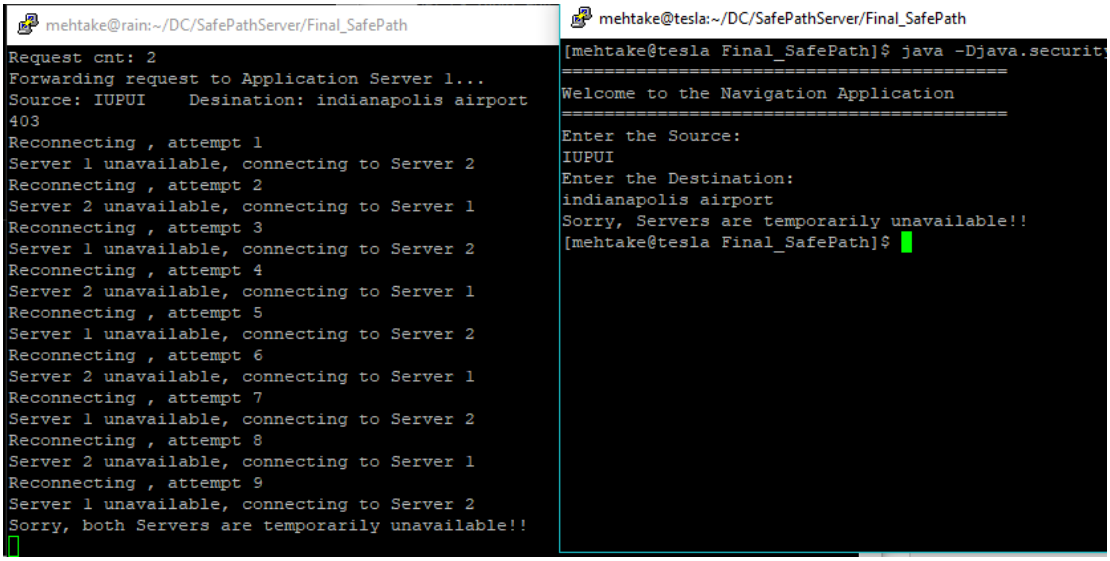
3. Failure Scenarios- One of the Server fails

When either of the server is down, the load balancer will send the requests to the other available server. And get the safest route for given request.



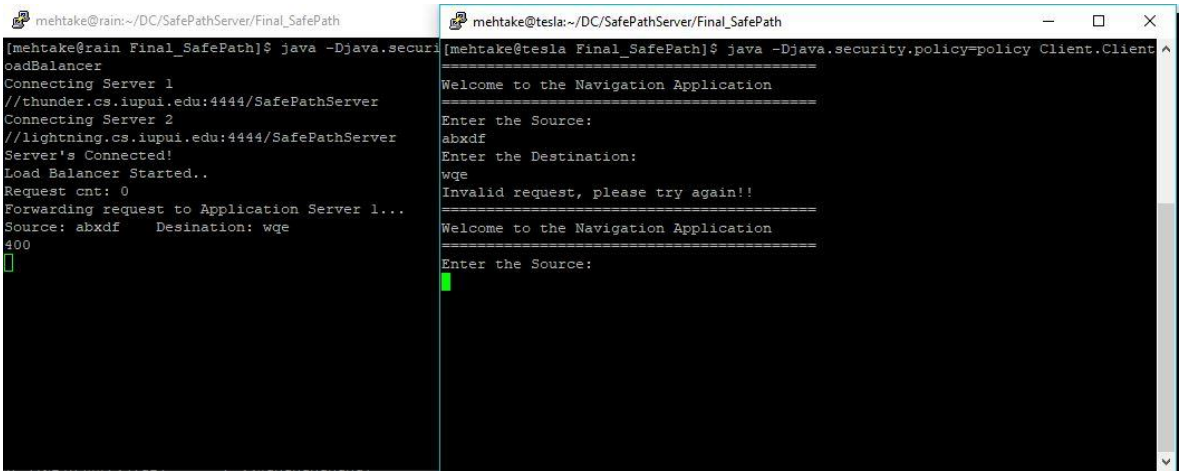
4. Failure Scenario- Both the server fails

When both the application servers are down, the load balancer will try to connect 10 times to each of the server. And if all the attempt fails than the load balancer will inform the client that the servers are unavailable.



5. Invalid Request

In the following screenshot you can see appropriate message is displayed to user when the user tries to enter any random text which is not appropriate. The application well handles such scenarios appropriately.



7. Analysis

The model is executed for number of requests and the time taken by each step of the safest route finding process is noted. The average time taken (in milliseconds) by these steps is mentioned in the below table. The major time involved in the process is to make database call and fetch all the incidents along the path. And the time to get the incidents on routes is increasing due as number of requests are increased. This is due to only single connection is maintained for the database. (Read – write locks on CRUD operation)

Table 1 Result Analysis

No of Request	Google API Call Time	DB call Time	Algo. running Time	E2E Time
50	994.56	74178.14	236.54	75409.24
250	105.39	358747.80	227.06	359080.2
500	78.90	721516.57	225.89	721821.4

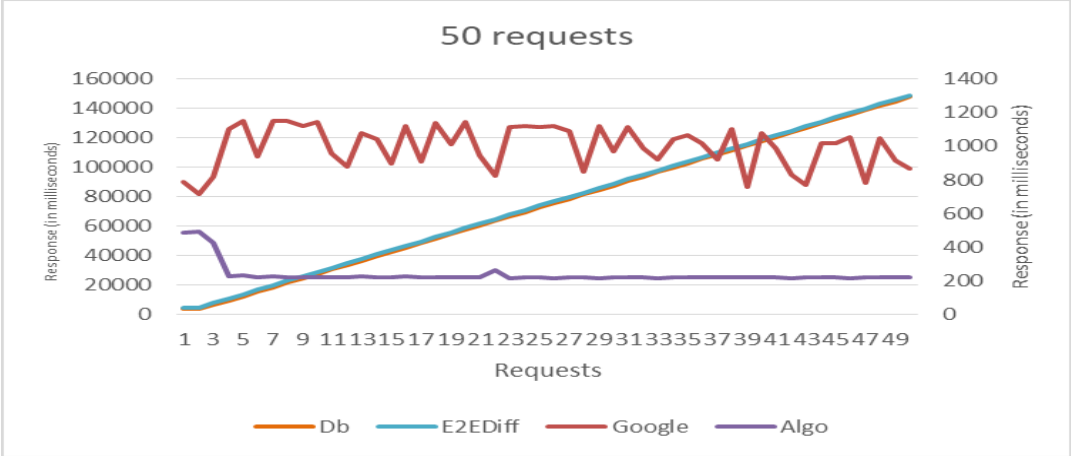


Figure 5 Analysis for 50 Request

The graph shows the time taken by each request for different components of the safe route determining process for 50 requests. The DB call and the E2EDiff uses the primary vertical axis and the Google API call and Algorithm processing time uses the secondary vertical axis. It can be evaluated from the graph that as number of requests are increased the E2E time to process the same is also increasing. The average time for API call is observed to be little high than other 2 cases.

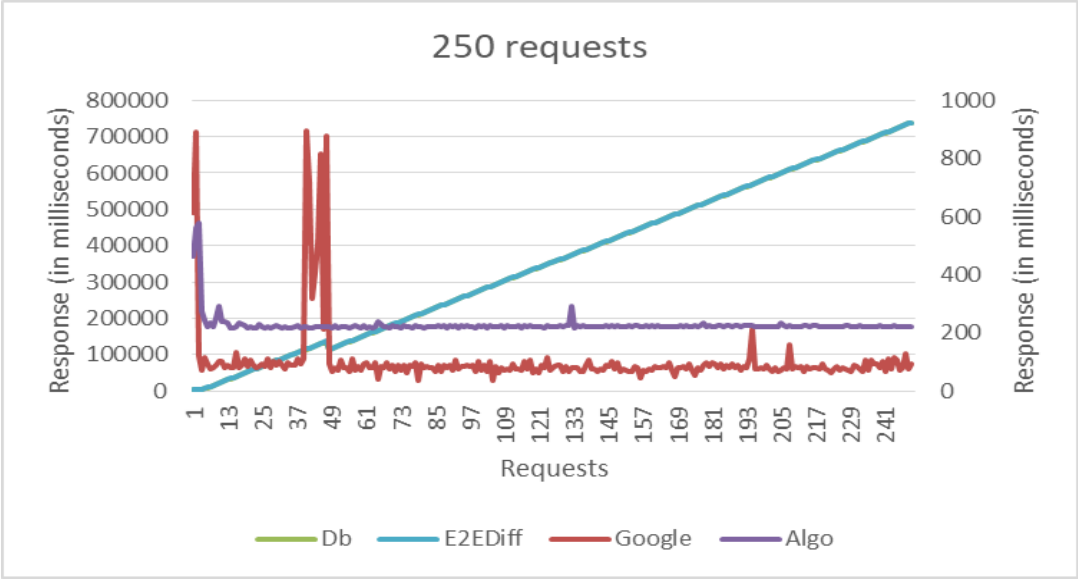


Figure 6 Analysis for 250 Request

The graph shows the time taken by each request for different components of the safe route determining process for 250 requests. The DB call and the E2EDiff uses the primary vertical axis and the Google API call and Algorithm processing time uses the secondary vertical axis. DB call time and the E2E time are almost equivalent to each other as the other 2 components takes very less time compare to DB call. The few spikes in the graph might be due to some network gap but the average time for the API call remain low.

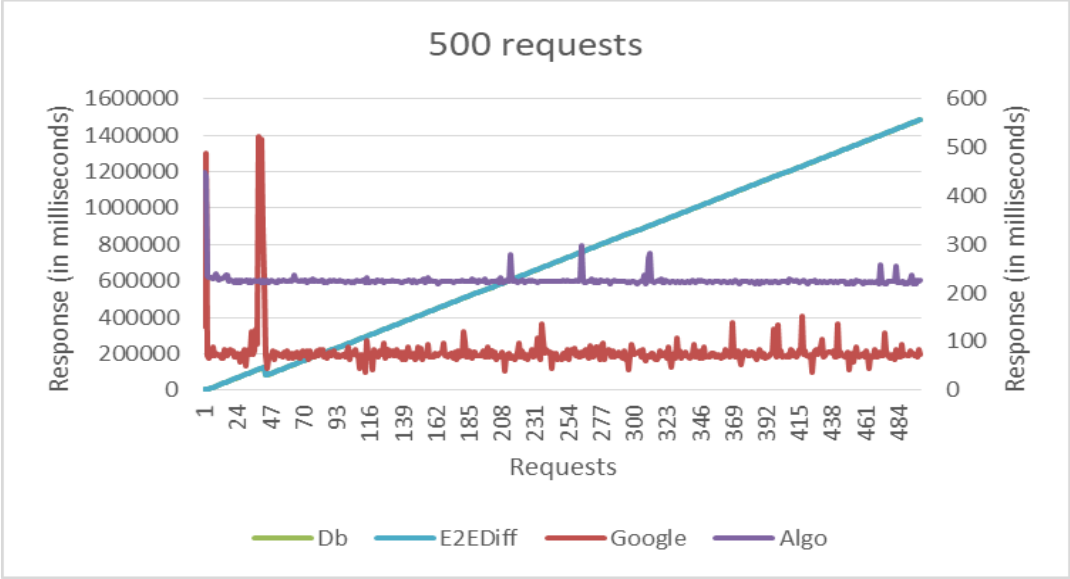


Figure 7 Analysis for 500 Request

The graph shows the time taken by each request for different components of the safe route determining process for 500 requests. The DB call and the E2EDiff uses the primary vertical axis and the Google API call and Algorithm processing time uses the secondary vertical axis. And the positive slop shows as the number of requests plays an important part in the performance of the application. It can be inferred from the above graphs that most time is taken in I/O operations of database in the entire application.

8. Critique

It is very difficult to build a perfect distributed system at the start of development. When we were trying to come up with a solution for this problem our main focus here was to figure out how we could trouble shoot this problem rather than building a perfect model. So, most of the design choices were driven by who is going to use and how this can be useful. Our current system has lot of manual steps involved in it to use the application. To eliminate these barriers following are things which we would like to go back and change all over again-

1. Change the communication paradigm between various components to webservices

In the current system we are using JavaRMI [3] to communicate between the client and the server-side components (Load Balancer, application server and database server). Every time to run the application it takes time to setup the environment and start the registry before using the application. This is an additional over head in the application. If we use webservices in our application, then it will make use of the existing Internet technology to communicate with each other and we can write the code which will take care of all these things. Data passing and managing data will be lot easier. With the implementation of webservices; this feature which will take care of run-time crime incidents happening near the current location of user will be easier. This will allow us to record all the live events and notify all the other users on the same route about the incident.

2. Database calls are expensive

We tried running our application for 1000 concurrent client threads. In this case for each thread a new database connection is established and then the query is executed. Although, our application server is able to handle these many requests concurrently, the database server takes a long time to traverse through the database to find the matching incidents for each step. This delay caused is affecting the total turnaround time of our system. In our case it is the size of data that is affecting along with the hardware configuration of the machine on which the DB is running. This problem can be resolved if we migrate from relational database to big data technologies that will help in advancing the lookup process.

3. Time of route and weather conditions for route calculation

Current system computes the path based on distance and the crime cost of the entire route. We are not considering other parameters like the time of commute and weather conditions along the route. These parameters can be taken into account to compute the path for better results.

9. Conclusion

In the current application we were able to calculate the total crime cost of the entire route; considering the number of incidents within 1 mile of the current GPS coordinates and thus return to the user safest and shortest path based on the cost's values. This backend model can act as an add-on to the existing navigation system to make the commute lot easier and safer for the users. This can ultimately make the cities more livable.

10. Future Work

1. Integrate it with any navigation application for its wide use.
2. Include the feature to run time record the crime incidents and navigate the users accordingly
3. Migrate from relational database to Big Data

ACKNOWLEDGEMENT

We express our gratitude to Dr. Rajeev Raje, who provided us with all the guidance and encouragement. We also would like to deeply express our sincere gratitude to Saurabh Pandey for providing us the needed assistance, detailed suggestions and encouragement to do the project

References

[1] J. C. R. George Mohler, "Improving social harm indices with a modulated Hawkes Process".

[2] "Direction API," [Online]. Available: https://developers.google.com/maps/documentation/directions/start.

[3] [Online]. Available: https://www.quora.com/How-does-the-algorithm-of-Google-Maps-work.

[4] "RMI," [Online]. Available: https://docs.oracle.com/javase/tutorial/rmi/index.html.