Detailed Design of Google Maps

Let's look into the detailed design of the maps system.

We'll cover the following

In this lesson, we'll discuss our detailed design by answering the following questions:

1. How do user requests benefit from segments?
2. How do we improve the user experience by increasing the accuracy of ETAs?

Segment setup and request handling

This section will describe how the segment data is stored in the database and how user requests are handled using the already stored data.

Starting with the storage schema, we discuss how the segments are added and hosted on the servers and also how the user requests are processed.

Storage schema

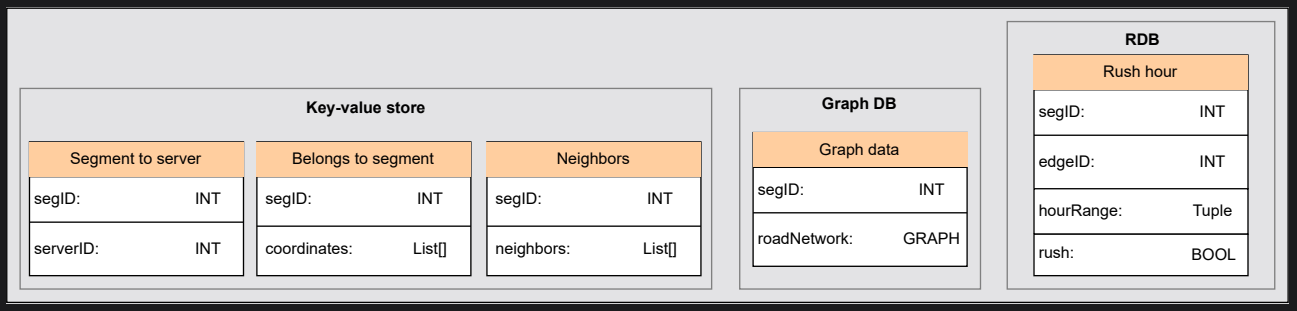
We store the following information for each segment:

Key-value store:

- The segment's `ID`.
- The `serverID` on which the segment is hosted.
- In reality, each segment is a polygon, so we store boundary `coordinates` (latitude/longitude), possibly as a list.
- A list of segment IDs of the `neighbors` segments.

Graph database

- The road network inside the segment in the form of a `graph`.



Storage schema

Relational DB

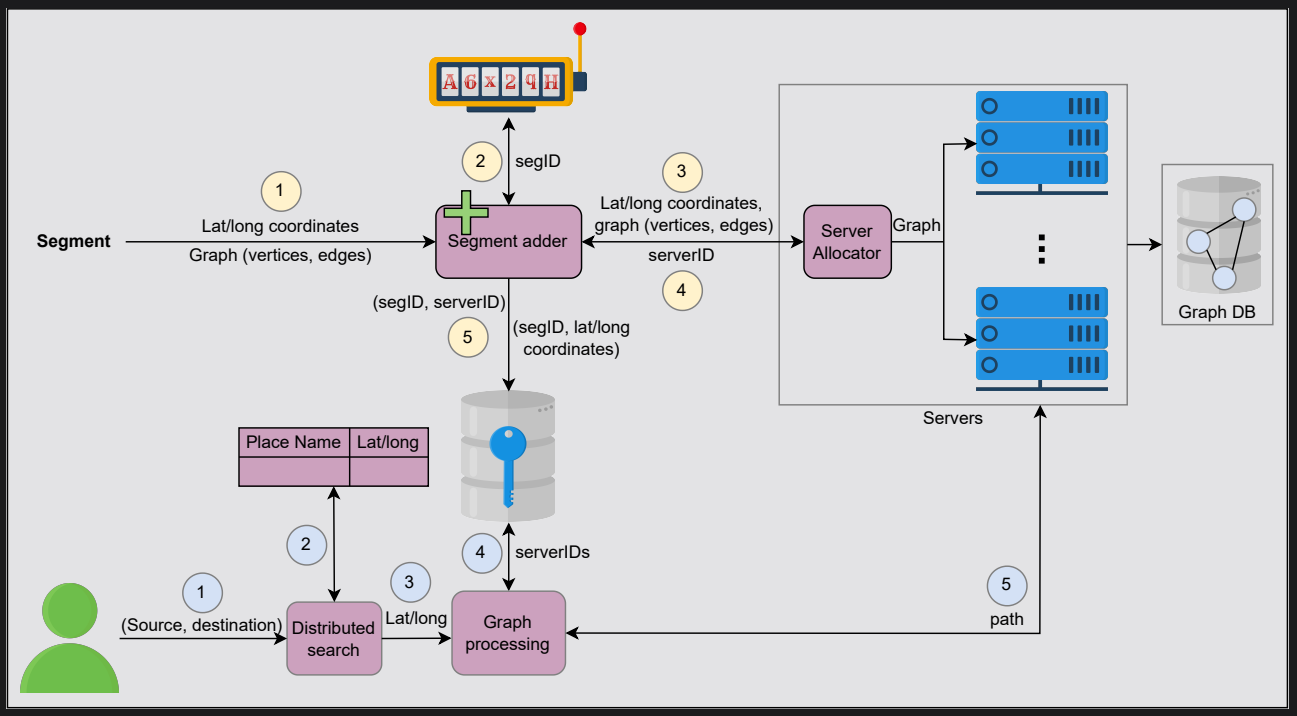
We store the information to determine whether, at a particular hour of the day, the roads are congested. This later helps us decide whether or not to update the graph (weights) based on the live data.

- `edgeID` identifies the edge.
- `hourRange` tells us which hour of the day it is when there are typical road conditions (non-rush hour) on the road.
- `rush` is a Boolean value that depicts whether there is congestion or not on a specific road at a specific time.

> **Note:** `segID`, `serverID`, and `edgeID` are unique IDs generated by a uniqueID generator (see the Sequencer lessons for details).

Design

The following illustration consists of two workflows. One adds segments to the map and hosts them on the severs, while the other shows how the user request to find a path between two points is processed.



Workflow 1) Adding segments (in yellow), and workflow 2) Processing user requests (in blue)

Add segment

1. Each segment has its latitude/longitude boundary coordinates and the graph of its road network.
2. The segment adder processes the request to add the segment along with the segment information. The segment adder assigns a unique ID to each segment using a unique ID generator system.
3. After assigning the ID to the segment, the segment adder forwards the segment information to the server allocator.
4. The server allocator assigns a server to the segment, hosts that segment graph on that server, and returns the `serverID` to the segment adder.
5. After the segment is assigned to the server, the segment adder stores the segment to server mapping in the key-value store. It helps in finding the appropriate servers to process user requests. It also stores each segment boundary latitude/longitude coordinates in a separate key-value object.

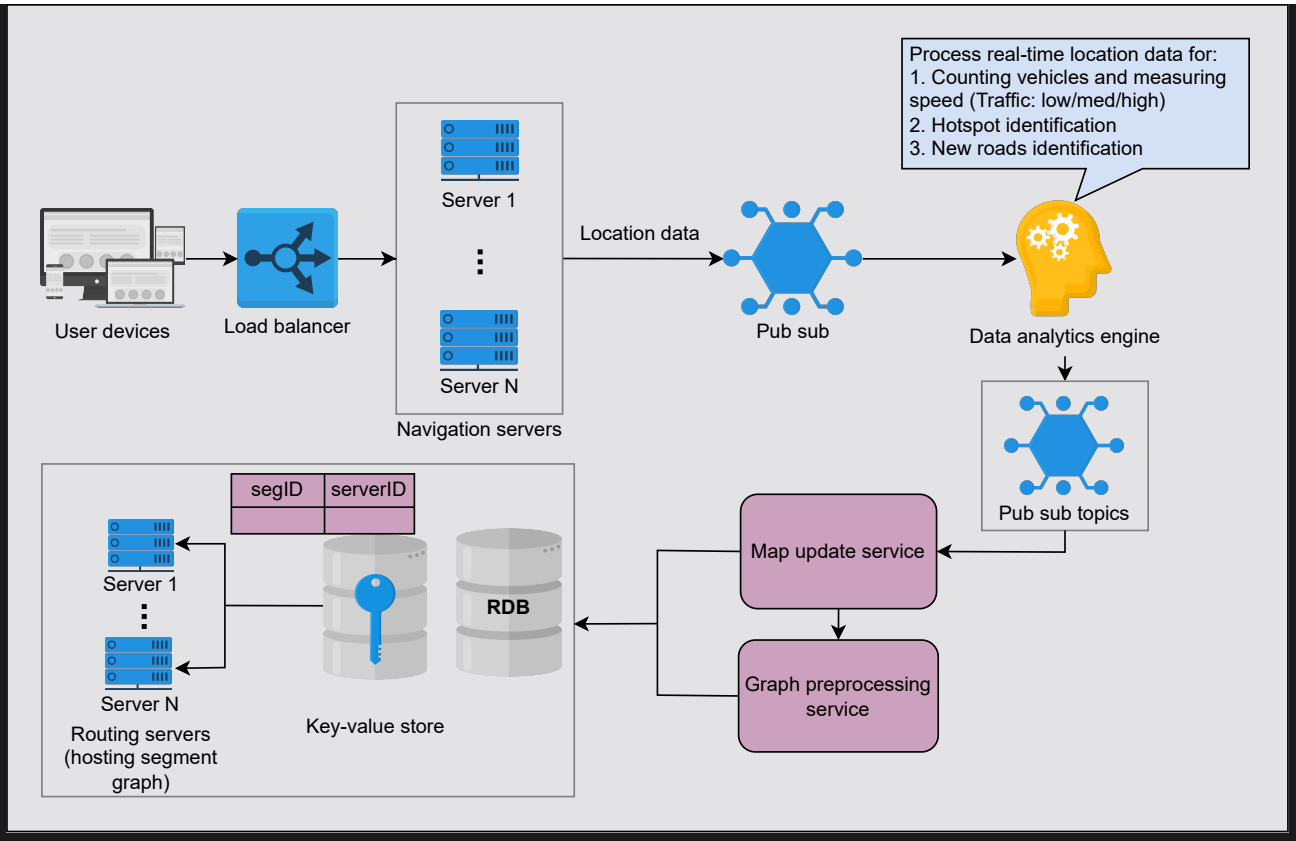
Handle the user's request

1. The user provides the source and the destination so that our service can find the path between them.
2. The latitude and longitude of the source and the destination are determined through a distributed search.
3. The latitude/longitude for the source and the destination are passed to the graph processing service that finds the segments in which the source and the destination latitude/longitude lie.
4. After finding the segment IDs, the graph processing service finds the servers that are hosting these segments from the key-value store.
5. The graph processing service connects to the relevant servers to find the shortest path between the source and the destination. If the source and the destination belong to the same segment, the graph processing service returns the shortest path by running the query only on a single segment. Otherwise, it will connect the segments from different servers, as we have seen in the previous lesson.

Improve estimations using live data

This section describes how we can improve the ETA estimation accuracy using live data. If we have a sequence of location data for different devices, we can find movement patterns and perform analytics to predict various factors that may influence a user's trip. Google Maps uses a combination of GPS, Wi-Fi, and cell towers to track users' locations. To collect this data, our maps system servers need to have a persistent connection with all the devices that have their location turned on. Below, we discuss the tools, techniques, and components involved in the process of improving estimations using live data.

- **WebSocket** is a communication protocol that allows users and servers to have a two-way, interactive communication session. This helps in the real-time transfer of data between user and server.
- The **load balancer** balances the connection load between different servers since there is a limit on the number of WebSocket connections per server. It connects some devices to server 1, some to server 2, and so on.



Performing analytics on the live location data to keep the map up-to-date and also improve ETA estimations

- A **pub-sub** system collects the location data streams (device, time, location) from all servers. The location data from pub-sub is read by a **data analytics engine** like Apache Spark. The data analytics engine uses data science techniques—such as machine learning, clustering, and so on—to measure and predict traffic on the roads, identify gatherings, hotspots, events, find out new roads, and so on. These analytics help our system improve ETAs.

> **Note:** The amount of traffic, road conditions, and hotspots directly affect average travel speed, which ultimately affects users' ETAs.

- The data analytics engine publishes the analytics data to a new **pub-sub topic**.
- The **map update service** listens to the updates from the pub-sub topic fo the analytics. It updates the segment graphs if there is a new road identified or if there is a change in the weight (`average speed (traffic,`

`road condition`) on the edges of the graph. Depending on the location, we know which segment the update belongs to. We find the **routing server** on which that segment is placed from the key-value store and update the graph on that server.

Point to Ponder

Question

Many traffic conditions are transitory (like stopping at a signal), so updating the graph very often wouldn't scale well because it requires excessive processing. What could be the solution to this problem?

Show Answer

- The **graph preprocessing service** recalculates the new paths on the updated segment graph. We've seen how the paths are updated continuously in the background based on the live data.

We've learned how the segments work, how a user finds the location between two points, and how the ETA's accuracy is improved by utilizing the live location data.