Back To Module Home

Design Problems

0% completed

RESHADED Approach for System Design

Design YouTube

Design Quora

Design Google Maps

- System Design: Google Maps
- Requirements of Google Maps' Design
- Design of Google Maps
- Challenges of Google Maps' Design

Detailed Design of Google Maps

Evaluation of Google Maps' Design

Design a Proximity Service / Yelp

Design Uber

Design Twitter

Design Newsfeed System

Design Instagram

Design a URL Shortening Service / TinyURL

Design a Web Crawler

Design WhatsApp

Design Typeahead Suggestion

Design a Collaborative Document Editing Service / Google Docs

Conclusion

Mark Module as Completed

Evaluation of Google Maps' Design

Let’s look at how our map design meets the requirements.

We'll cover the following

Let’s see how the system we designed will handle millions of queries per second, ensuring a fast response time.

Availability#

With a large road network graph hosted on a single server, we ran into these issues:

- We couldn’t process user queries, since it was impossible to load such a large graph into the memory, making the system unavailable to the users.
- It wasn’t possible to make a persistent two-way connection (for navigation) between the server and millions of users per second.
- It was also a single point of failure.

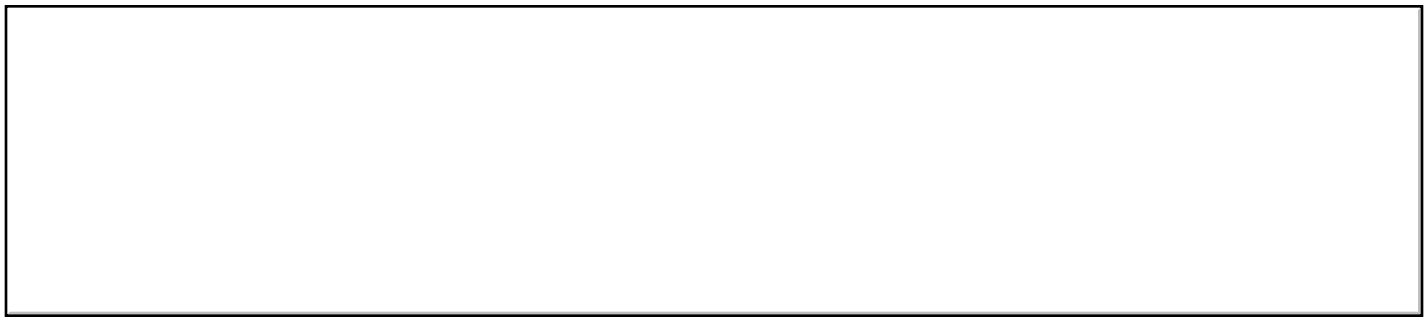
We solved the above problems by dividing the world into small segments. Each small segment consists of a graph that can be easily loaded into a server’s memory. With

segments, we completed these objectives:

- We hosted each segment on a separate server, mitigating the issue of loading a large, global graph.
- The load balancer divides the request load across different segment servers depending on the user's area of search. It mitigates the issue of putting the burden on a single server, which was affecting the system's availability.
- We didn't discuss replication, but we can replicate each segment, which will help deal with a segment server as a single point of failure and distribute the request load for a segment to replicas.

> **Note:** Google Maps uses lazy loading of data, putting less burden on the system, which improves availability.

Lazy loading reduces initial load time by reducing the amount of content to load, saves bandwidth by delivering content to users when needed, and preserves server and client resources by rendering only some of the content.



Non-functional requirements fulfilled by our maps system

Scalability#

We scaled our system for large road networks. Scalability can be seen in two ways:

- The ability of the system to handle an increasing amount of user requests.

- The ability of the system to work with more data (segments).

We divided the world into small segments. Each segment is hosted on a different server in the distributed system. The user requests for different routes are served from the different segment servers. In this way, we can serve millions of user requests.

> **Note:** It wouldn't have been possible to serve millions of user requests if we had a single large graph spanning the whole road network. There would have been memory issues loading and processing a huge graph.

We can also add more segments easily because we don't have to change the complete graph. We can further improve scalability by non-uniformly selecting the size of a segment—selecting smaller segment sizes in densely connected areas and bigger segments for the outskirts.

Smaller response times#

We're running the user requests on small subgraphs. Processing a small subgraph of hundreds of vertices is far faster than a graph of millions of vertices. We can cache the processed small subgraph in the main memory and quickly respond to user requests. This is how our system responds to the user in less time.

There's another aspect that helps our system to respond quickly, and that is keeping the segment information in the key-value store. The key-value store helps different services to get the required information quickly.

- The graph processing service checks for the relevant segments in which the source and the destination latitude/longitude lie by querying the key-value store for the `segmentID` values.
- For load-balancing user requests among different segment servers, the key-value store is queried for the `serverID` against the segment on which the graph processing should run for a specific request.

Accuracy#

Besides the road data we had initially, we also captured the live location data of users, on which we performed analytics using data science techniques. Our system improves the accuracy of the results (path, ETA) using these analytics. Based on the analytics of the traffic patterns, the maps are updated, and the routes and ETA estimations are improved.

Meeting Non-functional Requirements

Requirements	Techniques
Availability	• Process user queries on small graphs (segments). • Load balance requests across different segment servers. • Replicate segment servers.
Scalability	• Partition the large graphs into small graphs to ease segment addition • Host the segments on different servers to enable serving more querie second.
Less response time	• Cache the processed graphs. • Use a key-value store to quickly get the required information.
Accuracy	• Collect live data. • Perform data analytics.

Conclusion#

Google Maps is one of the most widely used applications in the world, where users find the shortest route between two locations. A map system models the road network with a graph data structure. To find the route, the shortest path algorithm runs over the graph. We've seen scalability issues with a large graph of the road network. We solved the problem by splitting the world into small segments. Each segment consists of a small graph that can be loaded into the memory to find the paths quickly. We've also seen that the estimated time of arrival can be improved by analyzing the live location data.

Detailed Design of Google Maps

Next

System Design: Yelp

Mark as Completed

Report an Issue