Google maps

Step 1: understand problem and establish design scope

qu: daily active user? ans: 1 bn DAU

qu: fature to focus on? ans: location update, navigation, estimated time arrival, map rendering

qu: how large is road data? do we have access to it?
ans: yes, we have road data from different source, TBs of raw data.

qu: system take traffic conditions into consideration?
ans: yes, traffic condition are important for accurate time estimation

qu: different travel modes? ans: yes, walking, drivinf, bus

qu: should support multi-stop directions. ans: not for now

qu: how about business place and photos? how many photos are we expecting?
ans: not for now

functional requirements:

1. user location update
2. navigation service, including ETA
3. map rendering

nonfunctional requirements

1. accuracy - user shouldn't give wrong direction
2. smooth navigation - client should experience smooth map rendering
3. data and battery usage - as little data and battery as possible
4. general availability and scalability requirements

map 101 positioning system

1. lat (latitude) - how far north or south we are
2. long (longitude) - how far east or west we are

going from 3d to 2d - map projection

geocoding - convert address to geographic coordinates (lat, long) - method called interpolation

geohashing - encoding system to encode geo area into short string of letter and digits, each grip represent of number between 0-3 - use for map tiling

map rendering - world divided into different set of tiles at different zoom levels - provide right level of map details

road data processing for navigation

most routing algo are variation of Dijkstra or A pathfinding algo. These work on graph data structure (as node and graph edge).

use geohash to divide world into small section - each route tile hold reference to all other tiles it connects to.

hierarchical routing tiles

3 sets of routing tiles with different details level

1. most detailed - local roads
2. bigger tiles - aeterial roads connect district together
3. highest level - cover major highways and states together

<u>Step 2: back of the envelope estimation</u>

1. 1 foot = 0.3 meters
2. 1 km = 0.62 miles
3. 1 km = 1000 meters

storage use

1. **map of the world** - detailed calculation shown below
    1. eg at zoom level 21 - 4.3 trillion tiles - each is 256*256 pixel compressed PNG img, with size = 100 kb → entire set storage = 4.4 trillion * 100 kb = 440 PB
    2. 90% of world surface is ocean, desert etc → remaining size = 44-88 PB → avg 50 PB for now
    3. lower zoom level - zoom size drop by half → total reduction = 4x → 60 PB, so total rough estimate = 100 PB
2. metadata - skip for now, since metadata for date is very small and we can skip it
3. road info - transform routing data into routing tile

server throughput

1. 2 types of request - initiate navigation, and location update request. Location data used by downstream service eg input to live traffic data
2. eg 1 bn DAU → each navigation = 35 mins per week → j35 bn min per week → 5 bn min per day.
3. send GPS coordinate every second - 300 bn req per day → 5 mn mins*60 = 3 mn QPS, but we don't need to send update every sec, it depends on how fast the user is moving, eg if user is stuck in traffic → send batch in much lower rate eg every 15 secs → QPS reduce to 3 mn/15 = 200,000

Step 3: Propose HLD

1. get figure 7 HLD
2. HLD support 3 service - location service, navigation service, map rendering.
3. **location service**
    1. recod user location update: mobile user → lB → location service → user location DB
    2. call client to send location update every t seconds
    3. adv: use data to monitor live traffic, detect new or closed road, provide accurate ETA estimate to user and reroute around traffic
    4. location history cna be send in batch → but still we have high write volume → use db eg **Cassandra.**
    5. also need to log location data using stream process engine eg: **kafka**
    6. good protocol fit - HTTP

navigation service

1. find reasonably fast route from A to B, accuracy is critical
2. user send HTTP request through LB, with origin and dest as parameter

GET /v1/nav?origin=1355+market+street,SF&destination=Disneyland

map rendering

1. instead of saving entire data, fetch data on demand based on client location and zoom level
2. option1: server build map tile on fly based on client location and zoom level. adv: huge load on cluster to generate every map on fly. can't take adv of caching either
3. option2: user make HTTP req to fetch tile from CDN → if CDN not yet served that tile, fetch details from server, cache locally and return to user. on subsequent request, return cache copy to same or diff user (more scalable and performance approach)
    1. data storage in these → assume 5 bn min of navigation per day → 5 bn * 1.25 MB = 6.25 MB of map data per sec. assume we have 200 point of presend center to store CDN → each POP need 62500/200 per second.

    2. NOTE: since we encode geohash per grid, it is efficient to go for client location and zoom level to geohash map to tile to getch CDN
4. option3: instead of hardcode client side algo to convert lat/long pair and zoom level, introduce intermediary service which will construct tile URL based on input
    1. user call map tile service to fetch tile URL → req sent to LB → LB forward req to map tile service → service takes client location and zoom level as input and return 9 URL of tile to client (tile to render and 8 sorrounding tile) → client download tile from CDN

Step 4: Design deep dive

Data model

routing tiles

1. run routing tile processing service (pipeline) periodically to transform dataset into routing tile. we get 3 tile set with different resolution (each with list of graph nodes, and edge representing intersection and road within area covered by tile
2. Most graph data represent as adj list, store these tiles in S3 and cache . Use performance software package to serializer adj list to binary file. Organize by geohash, for fast lookup mechanism.

user location data

1. To update road data and routing tile, to build db of live and historical data. Use write-heavy workload eg Cassandra.

Geocoding database

1. Db store place and their corresponding lat/long pair.
2. KV db eg Redis for high read, in frequent writes

precomputed image of world map

device ask for particular areas map - get nearby road and compute image that represent that area all related road and details.

- compute these details once with different zoom level and cache the image, store them in CDN, backed by cloud storage eg S3

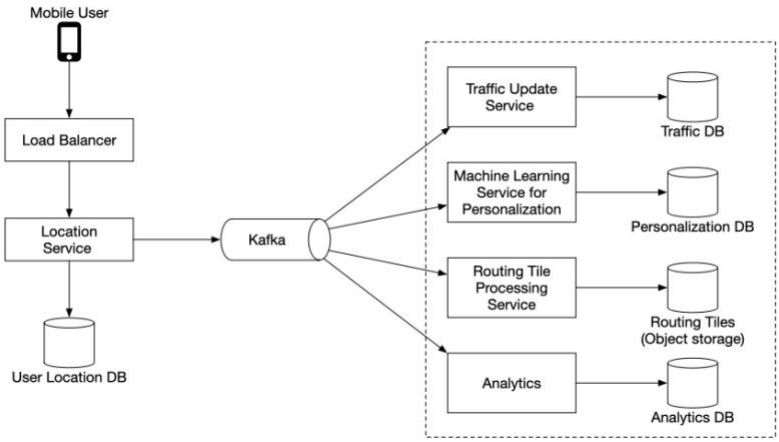
Services

location service

mobile user - load balancer - location service - user location DB

given 1 mn location updates - No SQL KV db - prioritize availability and partition tolerance over consistency - **Cassandra is good fit**

key = user_id, timestamp; value= lat/long pair (user_id as the partition key) use to improve accuracy to detect new and recently closed roads - log this information into Kafka



rendering map -

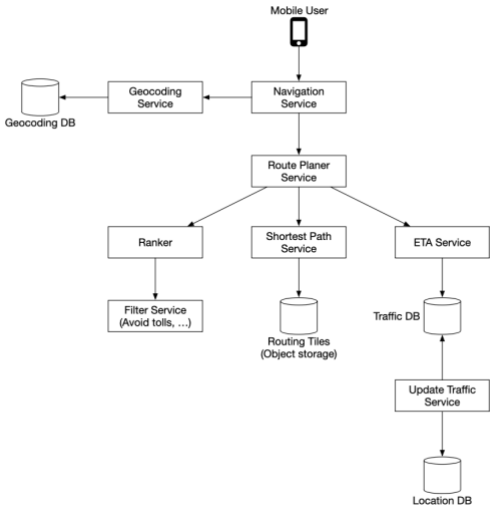
precomputed tiles: different set of precomputed map tiles at various distinct zoom level to provide appropriate map detail.

Level 0 - most zoom out level - entire map represent single level = 256*256

with each increment, number of tiles double in both north-south direction.

optimization - with development and implementation of WebGL - change design from sending image over network, to send vector information instead. -> compress data much better than image. Also vector tiles provide better zooming experience. Clint can scale each component smoothly and with better zooming experience.

navigation service



geocoding service - resolve address to location of lat/long

eg

https://maps.googleapis.com/maps/api/geocode/json?address=1600+Amphitheatre+Parkway,+Mountain+View,+CA

response:


```
{
  "results" : [
    {
      "formatted_address" : "1600 Amphitheatre Parkway, Mountain View, CA 94043, USA",
      "geometry" : {
        "location" : {
          "lat" : 37.4224764,
          "lng" : -122.0842499
        },
        "location_type" : "ROOFTOP",
        "viewport" : {
          "northeast" : {
            "lat" : 37.4238253802915,
            "lng" : -122.0829009197085
          },
          "southwest" : {
            "lat" : 37.4211274197085,
            "lng" : -122.0855988802915
          }
        }
      },
      "place_id" : "ChIJ2eUgeAK6j4ARbn5u_wAGqWA",
      "plus_code": {
        "compound_code": "CWC8+W5 Mountain View, California, United States",
        "global_code": "849VCWC8+W5"
      },
      "types" : [ "street_address" ]
    }
  ],
  "status" : "OK"
}
```


route planner service - suggest route with optimize time travel based on current traffic and road condition

shortest-path service - receive origin and destination in lat/long pair and return top-k shortest path without considering traffic and road condition

algorithm - receive origin and destination in lat/lang air - convert into geohash - load start and end routing tile.

- algo start traverse graph data structure, hydrate additional neighbor tile from object store, continue to expand its search until a set of best route is found.

ETA service - route planner call ETA service for each route and get time estimate. It use machine learning to predict ETA on current traffic and historical data

ranker service - once gt ETA, route planner pass this info to ranker service to apply possible filter defined by user (eg avoid toll, avoid freeway) - and give top K result

updater service - tap into kafkaesque stream and async update all db eg routing db (transform db with newly found data into updated tile set) and traffic db (extract traffic condition from stream sent by live traffic db), thus enable ETA more accurate

Delivery protocol

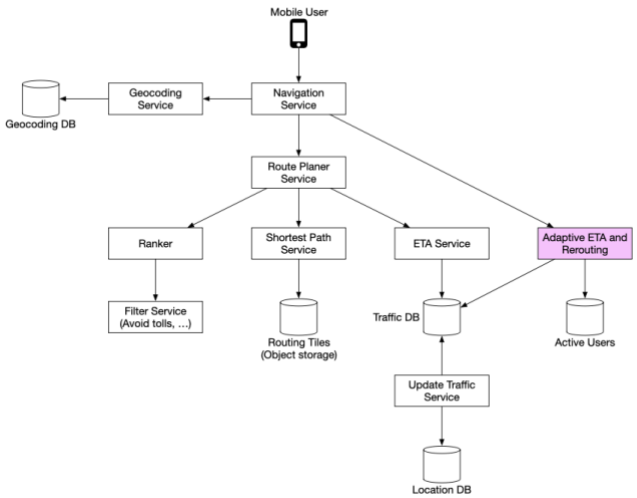
during navigation, route condition may change - some delivery protocol include mobile push condition, long polling, web socket, server sent event

mobile push notification - not great because of limited payload, as not supported web app

web socket - better option than long polling - very light footprint on server

mainly choose is web socket and SSE - web socket is better because bidirectional communication and feature like last mile delivery might require bidirectional real time communication

Final design



Chapter summary