The document "System Design: Google Maps - Grokking Modern System Design Interview for Engineers & Managers" provides a comprehensive overview of designing a system like Google Maps. Here's a detailed summary of the content:

### **Introduction**

The document is part of a course on system design interviews for engineers and managers, focusing on the design of Google Maps. It outlines the importance of understanding key system design concepts, including abstractions, non-functional system characteristics, and essential building blocks such as DNS, load balancers, databases, CDNs, and distributed monitoring.

### **Importance of Maps Services**

Maps services like Google Maps are essential for various modern applications, including ride-hailing services, autonomous vehicles, and logistics. They help users find the best possible paths to their destinations, considering factors like vehicle type, distance, and travel time. The document emphasizes the widespread use of Google Maps by individuals and businesses, highlighting its role in reducing travel time and costs, providing estimated time of arrival (ETA), and optimizing routes based on real-time traffic data.

### **Key Use Cases**

1. **Individuals**: Use maps to find locations and directions quickly, saving time and travel costs.
2. **Businesses**: Use maps for routing and logistics to minimize delivery times and costs.
3. **Applications**: Integrate maps for navigation in services like Waymo's self-driving cars and Uber.

### **Design Lessons**

The design of Google Maps is divided into five lessons:

1. **Requirements**:
   * **Functional Requirements**: Accurate mapping, real-time traffic updates, route optimization, and integration with other services.
   * **Non-Functional Requirements**: Scalability, reliability, low latency, and high availability.
   * **Challenges**: Handling large amounts of data, real-time processing, and ensuring data accuracy.
   * **Resource Estimation**: Servers and bandwidth needed to serve millions of users.
2. **Design**:
   * **High-Level Design**: Overview of the system's architecture and components.
   * **API Design**: Interfaces for interaction with the system.
   * **Services and Workflow**: Detailed description of how different services interact within the system.
3. **Meeting Challenges**:
   * **Data Handling**: Strategies for managing and processing large datasets efficiently.
   * **Real-Time Processing**: Techniques to ensure timely updates and responses.
   * **Accuracy and Reliability**: Methods to maintain data accuracy and system reliability.
4. **Detailed Design**:
   * **Storage Schema**: Design of the database schema for efficient data storage and retrieval.
   * **System Improvements**: Enhancements based on initial design challenges and solutions.
5. **Evaluation**:
   * **Performance Metrics**: How the designed system meets the specified requirements.
   * **Scalability and Reliability**: Assessment of the system's ability to handle increased load and maintain uptime.

### **Practical Applications**

The document further explores designing similar systems for different applications like YouTube, Quora, Yelp, Uber, Twitter, Instagram, and more, providing a broad perspective on system design principles applicable across various domains.

### **Conclusion**

The course concludes with a discussion on spectacular failures and lessons learned from them, emphasizing the importance of robust system design practices to avoid such pitfalls.

The document provides URLs to additional resources and a course certificate upon completion.

The document "Requirements of Google Maps' Design - Grokking Modern System Design Interview for Engineers & Managers" outlines the key requirements and considerations for designing a system like Google Maps. Here’s a detailed summary:

### **Functional Requirements**

1. **Identify Current Location**:
   * Users should be able to approximate their current location on the world map using latitude and longitude in decimal values.
2. **Recommend the Fastest Route**:
   * Given the source and destination (place names in text), the system should recommend the optimal route by distance and time, depending on the type of transportation.
3. **Give Directions**:
   * Once the user has chosen the route, the system should provide directions in text format, guiding the user to turn or continue in a specific direction to reach the destination.

### **Non-Functional Requirements**

1. **Availability**:
   * The system should be highly available to cater to the needs of both individuals and enterprise applications like Uber and Lyft.
2. **Scalability**:
   * The system should be scalable to handle millions of queries from different users simultaneously.
3. **Less Response Time**:
   * It should take no more than two to three seconds to calculate the ETA and the route, given the source and destination points.
4. **Accuracy**:
   * The predicted ETA should not deviate significantly from the actual travel time.

### **Challenges**

1. **Scalability**:
   * Serving millions of queries per second, given a graph with billions of nodes and edges spanning over 194 countries, requires robust scalability measures. Traditional path-finding algorithms like Dijkstra’s would not scale well for billions of users, necessitating alternative techniques.
2. **ETA Computation**:
   * Computing ETA is straightforward in ideal conditions with empty roads. However, real-world factors like traffic, road conditions, and incidents like construction or collisions need to be quantified and incorporated into the design, which is complex and challenging.

### **Resource Estimation**

1. **Number of Servers**:
   * Estimating the number of servers requires knowing the number of daily active users and the number of requests a single server can handle per second. Assumptions:
     + Daily active users: 32 million (about 1 billion monthly users).
     + Requests per second a server can handle: 8,000.
   * The total number of servers required is approximately 4,000.
2. **Storage Estimation**:
   * Google Maps has a one-time storage requirement, with road data from many countries already added, totaling over 20 petabytes as of 2022. The daily storage requirement is negligible since short-term changes in the road network are minimal compared to the full network data.
3. **Bandwidth Estimation**:
   * Estimating the bandwidth for incoming and outgoing traffic:
     + **Incoming Traffic**:
       - Maximum requests by a single user per day: 50.
       - Request size: 200 Bytes.
       - Total requests per second: 18,518.
       - Bandwidth required: 29.63 Mbps.
     + **Outgoing Traffic**:
       - Total requests per second: 18,518.
       - Response size: 2005 KB (2 MB for visual data and 5 KB for textual data).
       - Bandwidth required: 297.03 Gbps.

### **Building Blocks**

1. **Load Balancers**:
   * Distribute user requests among different servers and services.
2. **Databases**:
   * Store data in the form of a graph along with metadata information.
3. **Distributed Search**:
   * Search different places on the map.
4. **Pub-Sub System**:
   * Generate and respond to important events during navigation and notify corresponding services.
5. **Key-Value Store**:
   * Store metadata information.

### **Conclusion**

The document concludes by emphasizing the importance of these building blocks in designing a scalable, reliable, and efficient maps system like Google Maps. Further detailed design lessons will explore the high-level and API design, overcoming challenges, and evaluating the system’s performance.

This summary captures the essence of the document and highlights the key aspects of designing a system like Google Maps, focusing on both functional and non-functional requirements, challenges, resource estimation, and necessary building blocks.

The document "Design of Google Maps - Grokking Modern System Design Interview for Engineers & Managers" details the high-level design, components, workflow, and API design of a system like Google Maps. Here is a comprehensive summary:

### **High-Level Design**

The high-level design of a map system is divided into two sections:

1. **Components**
2. **Workflow**

### **Components**

The essential components in the design are:

1. **Location Finder**: This service finds and shows the user’s current location on the map.
2. **Route Finder**: This service identifies the best paths between two locations.
3. **Navigator**: This service tracks the user’s journey, updates directions, and sends notifications if the user deviates from the suggested route.
4. **GPS/Wi-Fi/Cellular Technology**: These technologies determine the user's ground position.
5. **Distributed Search**: Converts place names to latitude/longitude values and maintains an index of place names and their mappings.
6. **Area Search Service**: Coordinates between the distributed search and graph processing service to find the shortest path for a user query.
7. **Graph Processing Service**: Runs the shortest path algorithm on a specified area of the graph to determine the optimal path.
8. **Database**: Stores road data as a graph, using a graph database like DataStax Graph.
9. **Pub-Sub System**: Listens to various events and triggers other services accordingly, such as recalculating routes when users deviate from the path.
10. **Third-Party Road Data**: Collects and preprocesses road data from third-party resources.
11. **Graph Building Service**: Builds the graph from collected data.
12. **User**: Refers to the person or program using the map system.
13. **Load Balancer**: Distributes user requests among different servers and services.

### **Workflow**

The workflow is explained with the following steps:

1. **Entering Starting Point and Destination**:
   * The user enters their current location and destination.
   * The location finder service determines the current location using GPS, Wi-Fi, and cellular technology.
   * The user types the destination address, utilizing a Typeahead service for suggestions and avoiding spelling mistakes.
2. **Requesting Optimal Path**:
   * The user requests the optimal path.
   * The route finder service forwards the request to the area search service.
   * The area search service uses the distributed search to find latitude/longitude for the source and destination, then calculates the area on the map.
   * The area search service asks the graph processing service to find the optimal path within the specified area.
3. **Visualizing the Path and Getting Directions**:
   * The graph processing service finds the shortest path and returns it to the route finder service.
   * The user visualizes the path on the map, along with steps for navigation.
   * The navigator service tracks the user's path, updates their location, and provides turn-by-turn directions.
   * If the user deviates, an event is generated and sent to Kafka, which updates the area search service to recalculate the optimal path.

### **API Design**

The APIs for the maps service include:

1. **Show User’s Current Location**:
   * currLocation(location): Displays the user’s location on the map.
   * Parameters: location indicates whether the user’s location is on or off.
2. **Find Optimal Route**:
   * findRoute(source, destination, transport\_type): Finds the optimal route between two points.
   * Parameters:
     + source: Starting point in text format.
     + destination: Ending point in text format.
     + transport\_type (optional): Mode of transportation (e.g., bicycle, car, airplane).
3. **Get Directions**:
   * directions(curr\_location, steps): Provides text or sound alerts indicating where to turn and how far.
   * Parameters:
     + current\_location: Latitude/longitude of the user’s current location.
     + steps: Steps to reach the destination.

### **Summary**

The document provides a detailed overview of the components and workflow necessary to design a system like Google Maps, emphasizing the importance of real-time location tracking, optimal route finding, and user navigation. It also covers the necessary API functions to interact with the system effectively.

By following this structured design, the system can provide accurate, scalable, and reliable map services, essential for both individual users and enterprise applications.

The document "Challenges of Google Maps' Design - Grokking Modern System Design Interview for Engineers & Managers" outlines the key challenges and solutions in designing a system like Google Maps. Here’s a detailed summary:

### **Introduction**

The document addresses the major challenges faced when designing Google Maps, particularly focusing on scalability and ETA (Estimated Time of Arrival) computation.

### **Scalability**

Scalability involves efficiently processing a massive road network graph with billions of vertices and edges. The challenges include inefficient loading, updating, and performing computations on the graph, which increases query time for users. The proposed solution is to break down the large graph into smaller subgraphs or partitions, enabling parallel processing and querying. This segmentation approach reduces graph construction and query processing time significantly.

### **Segmentation**

* **Segments**: The globe is divided into small segments, each corresponding to a subgraph. These segments are small enough to be loaded, updated, and traversed easily in memory. For example, a city can be divided into hundreds of segments, each measuring 5×5 miles.
* **Coordinates**: Each segment has four coordinates (latitude and longitude) that help determine which segment a user is in.
* **Pathfinding within Segments**: Shortest path algorithms, like Dijkstra's algorithm, are used within each segment's graph to find the optimal paths. The results are stored in distributed storage to avoid recalculation and cache the most requested routes.

### **Connecting Segments**

* **Exit Points**: Each segment has boundary edges known as exit points that connect neighboring segments. The shortest path is calculated for these exit points and stored in the cache.
* **Inter-Segment Pathfinding**: For paths that span multiple segments, only the exit points and their cached information are considered, ignoring the internal graph of each segment. The aerial distance between source and destination is used to limit the number of segments involved, utilizing the haversine formula to calculate this distance.
* **Graph of Exit Points**: A graph is created using exit points as vertices and the calculated paths between them as edges. The shortest path algorithm is then applied to this graph to find routes between different segments.

### **ETA Computation**

* **Live Data Collection**: Live location data (userID, timestamp, latitude, longitude) is collected from the navigation service via a pub-sub system.
* **Traffic and Speed Calculation**: This data helps calculate and predict traffic patterns, average vehicle speeds, and traffic repetition intervals on different roads.
* **Accurate ETA**: The computed traffic information is used to provide more accurate ETA, accounting for high traffic times and specific road conditions.

### **Summary**

* **Scalability Solution**: The problem is divided into smaller, manageable segments, enabling efficient query processing on parts of the road network. For inter-segment queries, segments are connected through exit points, and the shortest path is calculated using cached information.
* **ETA Solution**: Accurate ETA is achieved by leveraging live data to understand and predict traffic patterns and average speeds.

By addressing these challenges through segmentation and real-time data analysis, Google Maps can offer scalable, accurate, and reliable map services, meeting both individual and enterprise needs.

The document "Detailed Design of Google Maps - Grokking Modern System Design Interview for Engineers & Managers" provides an in-depth look at the design considerations and implementation of Google Maps. Here's a comprehensive summary:

### **Detailed Design Overview**

The document covers:

1. Segment setup and request handling
2. Storage schema
3. Design workflows
4. Improving estimations using live data

### **Segment Setup and Request Handling**

#### **Storage Schema**

For each segment, the following information is stored:

* **Key-value store**:
  + Segment’s ID
  + ServerID on which the segment is hosted
  + Boundary coordinates (latitude/longitude) as a list
  + List of neighboring segment IDs
* **Graph database**:
  + The road network within the segment represented as a graph
* **Relational DB**:
  + Congestion information for different times of the day, helping in updating the graph weights based on live data

### **Design Workflows**

#### **Adding Segments**

1. **Boundary Coordinates and Graph**: Each segment is defined by its latitude/longitude boundary coordinates and the road network graph.
2. **Segment Adder**: Processes the request to add the segment, assigns a unique ID using a unique ID generator, and forwards the segment information to the server allocator.
3. **Server Allocator**: Assigns a server to the segment, hosts the segment graph, and returns the serverID.
4. **Key-Value Store**: Stores the segment-to-server mapping and boundary coordinates for quick lookup during user requests.

#### **Handling User Requests**

1. **Source and Destination**: The user provides the source and destination.
2. **Distributed Search**: Determines the latitude and longitude of the source and destination.
3. **Graph Processing Service**: Finds the segments containing the source and destination from the key-value store.
4. **Server Lookup**: Finds the servers hosting these segments.
5. **Pathfinding**: Connects to the relevant servers to find the shortest path. If the source and destination are within the same segment, the query runs on a single segment. Otherwise, it connects multiple segments.

### **Improving Estimations Using Live Data**

#### **Tools and Techniques**

* **WebSocket**: Enables real-time, two-way communication between users and servers.
* **Load Balancer**: Distributes WebSocket connections among multiple servers.
* **Pub-Sub System**: Collects location data streams (device, time, location) from all servers.
* **Data Analytics Engine (e.g., Apache Spark)**: Analyzes location data to measure and predict traffic, identify gatherings, hotspots, and new roads, and improve ETA estimations.

#### **Analytics Process**

1. **Data Collection**: Persistent connection with devices that have location services turned on.
2. **Traffic and Road Conditions**: Analyze the data to measure traffic, average speeds, and other factors affecting travel.
3. **Map Updates**: Updates segment graphs with new roads or changes in traffic conditions, identified through analytics.
4. **Routing Server Update**: The segment's routing server is updated with the new information from the analytics data.

### **Conclusion**

The detailed design of Google Maps involves segmenting the map into manageable sections, efficiently handling user requests, and continuously improving ETA accuracy using live data analytics. By leveraging real-time data and advanced analytics, Google Maps can provide accurate and reliable navigation services, adapting to changing traffic conditions and road networks.

The document "Evaluation of Google Maps' Design - Grokking Modern System Design Interview for Engineers & Managers" provides a detailed assessment of how the design of Google Maps meets various requirements, including availability, scalability, response time, and accuracy. Here is a comprehensive summary:

### **Evaluation Criteria**

The evaluation focuses on:

1. **Availability**
2. **Scalability**
3. **Response Time**
4. **Accuracy**

### **Availability**

Challenges and solutions related to availability include:

* **Initial Issues**: Hosting a large road network graph on a single server resulted in issues such as the inability to process user queries, difficulty maintaining persistent connections, and creating a single point of failure.
* **Solution**: Dividing the world into smaller segments, each segment consisting of a manageable graph that can be loaded into memory. This approach accomplished:
  + Hosting each segment on a separate server, avoiding the problem of loading a large global graph.
  + Using a load balancer to distribute requests across different segment servers based on the user's area of search.
  + Potential for replication of segments to handle server failures and distribute the request load.

**Lazy Loading**:

* Implemented to reduce the initial load time by delivering content to users as needed, saving bandwidth, and preserving server resources.

### **Scalability**

Scalability is addressed in two aspects:

1. **Handling Increased User Requests**: By dividing the world into small segments and hosting each on different servers, millions of user requests can be handled efficiently.
2. **Managing More Data (Segments)**:
   * Adding more segments without altering the entire graph.
   * Selecting smaller segments in densely connected areas and larger segments in less dense areas to optimize performance.

### **Response Time**

Strategies to achieve smaller response times:

* **Processing Small Subgraphs**: Running user requests on small subgraphs, which are faster to process than a large global graph.
* **Caching Processed Subgraphs**: Storing these subgraphs in main memory for quick access.
* **Key-Value Store**:
  + Quickly retrieves segmentID values to identify relevant segments for user requests.
  + Facilitates load balancing by quickly identifying serverID for the segment where graph processing should occur.

### **Accuracy**

Improvements in accuracy involve:

* **Live Data Collection**: Capturing live location data from users and performing analytics using data science techniques.
* **Traffic Pattern Analysis**: Updating maps and improving routes and ETA estimations based on analyzed traffic patterns.

### **Meeting Non-Functional Requirements**

**Techniques Employed**:

* **Availability**:
  + Processing user queries on small graphs.
  + Load balancing requests across segment servers.
  + Replicating segment servers.
* **Scalability**:
  + Partitioning large graphs into smaller ones.
  + Hosting segments on different servers to handle more queries.
* **Response Time**:
  + Caching processed graphs.
  + Using a key-value store for quick information retrieval.
* **Accuracy**:
  + Collecting live data.
  + Performing data analytics to update maps and improve ETA estimations.

### **Conclusion**

Google Maps effectively meets the criteria for a scalable, accurate, and responsive map service by:

* **Segmenting the Global Graph**: This approach mitigates scalability and memory issues associated with a large, monolithic graph.
* **Efficient Request Handling**: User requests are processed quickly by operating on smaller, manageable segments.
* **Real-Time Updates**: Utilizing live data to enhance accuracy and improve user experience.

By implementing these strategies, Google Maps provides a reliable and efficient navigation service that can handle millions of queries while ensuring fast response times and accurate route estimations.