## **Chat history**

[ChatGPT](https://chatgpt.com/)

[ChatGPT](https://chatgpt.com/)

### **Today**

1. [Designing Distributed Search System](https://chatgpt.com/c/2901ecc6-4e85-4f2c-a767-0a575692c8d5)
2. [Correcting Zigzag Traversal](https://chatgpt.com/c/4a7d1b52-4d10-41e5-aa62-b14432503762)

### **Previous 7 Days**

1. [Aiuta App Overview](https://chatgpt.com/c/e350a96f-83e9-4f39-9fae-e2bbb48d0313)
2. [Sequencer Design Overview](https://chatgpt.com/c/de94f9e1-d719-440e-8cd6-9be68d084402)
3. [Lady in Khwarazmian Empire Fashion](https://chatgpt.com/c/9e90b02d-e38f-4a5d-963f-997fbc9e9df0)
4. [Load Balancer Overview](https://chatgpt.com/c/6f18efd2-11b0-4b16-8717-b96015def87a)

### **Previous 30 Days**

1. [Positive Hook Type-B Explanation](https://chatgpt.com/c/a6871fe0-44fc-4f3e-9d34-34a7022884cf)
2. [Forex Trading Strategy Explanation](https://chatgpt.com/c/3e35e0eb-ab50-4a45-887d-bd8668ec4105)
3. [Forex Trading Hooks & NDS](https://chatgpt.com/c/3980fcc4-9cae-4e3d-aba8-bf9dbab3d6ba)
4. [Fashion brand name suggestions](https://chatgpt.com/c/7c9c162c-220a-4724-ae4c-d6b4d4dce396)
5. [Machine Learning Engineer Application](https://chatgpt.com/c/50c236e9-c185-455d-a145-a71d61174c4a)
6. [Cover Letter Refinement](https://chatgpt.com/c/8e6a3e0a-c765-4310-a9aa-2b5f0e65ee4d)
7. [Software Development Life Cycle Experience](https://chatgpt.com/c/d73b6483-0048-4dc8-bdfe-00e79487ac58)
8. [Stylish Woman in Embroidered Outfit](https://chatgpt.com/c/210efa10-bd8c-47fb-8a83-82bf71be9ec0)
9. [Woman in serene pose](https://chatgpt.com/c/7cb41311-a14e-431e-9cb2-ec4efc6ace6f)
10. [Hallway with intricate design](https://chatgpt.com/c/bff960bd-37e7-4fb9-9670-0851368cd868)
11. [Amazon SDE Test Overview](https://chatgpt.com/c/a58f5877-69a3-4935-bc80-c8b4a890b0a7)
12. [Hamster Kombat Coin Value.](https://chatgpt.com/c/02803877-28fe-4bbd-a6c3-d42b0e95560d)
13. [AI Engineer Resume Feedback](https://chatgpt.com/c/025a650b-be26-4fa4-8813-ca0374356aa4)
14. [Resume Review Request](https://chatgpt.com/c/7a2a2bfe-c2c7-401f-8ab2-5c4bb5566679)
15. [Dating Averages: 19-39](https://chatgpt.com/c/7e4d9d7e-9b95-4120-820f-53b65ffe88af)
16. [Time Tracker App - Requirements](https://chatgpt.com/c/1c009da8-4899-4b51-85b3-f3d6844971f2)
17. [Code Extension Exam Prep](https://chatgpt.com/c/1e1be9b3-4177-448f-8671-f0846a6a41bb)
18. [Web to PWA for Stores](https://chatgpt.com/c/b86839d8-bf35-4f0d-a578-c42907237ebc)
19. [Teaching Philosophy and Demo](https://chatgpt.com/c/42f1df4e-6211-417a-8548-120ac551ac78)
20. [Fashion at Iranian Historical Site](https://chatgpt.com/c/2d7aa858-f42d-406c-9cb3-b0a9c5cd64a7)
21. [Schedule Call Senior ML.](https://chatgpt.com/c/aacb026f-ec08-4713-8d82-88150cb5ed5f)
22. [Listen Carefully, Summarize Succinctly](https://chatgpt.com/c/db4a042d-bb22-44d0-b904-f448bc513611)

Add Team workspaceCollaborate on a Team plan

ChatGPT 4o

1. System Design\_ The Distributed Search.pdf

PDF

As an expert in system design, summarize the following document in great detail.

ChatGPT

The document "System Design: The Distributed Search" from the Grokking Modern System Design Interview for Engineers & Managers course provides a comprehensive guide on how to design a distributed search system. It is divided into several sections that cover the motivation, components, design, and evaluation of such a system.

### **Motivation for a Search System**

A search system is essential for quickly finding relevant content among vast amounts of data. Examples include websites with numerous pages, video platforms like YouTube with billions of videos, and the internet itself with countless web pages. Without a search system, users would struggle to find specific information efficiently.

### **What is a Search System?**

A search system is composed of three primary components:

1. **Crawler**: Fetches content and creates documents.
2. **Indexer**: Builds a searchable index from the fetched content.
3. **Searcher**: Responds to user queries by running the search query on the index created by the indexer.

### **Designing a Distributed Search System**

The design process is divided into five key lessons:

1. **Requirements**:
   * **Functional Requirements**: The system should handle a large volume of queries and return results quickly. It should support various types of searches, such as keyword search, phrase search, and fuzzy search.
   * **Non-functional Requirements**: The system must be scalable, highly available, and fault-tolerant. It should also ensure data consistency and handle varying loads efficiently.
   * **Resource Estimation**: This involves calculating the required number of servers, storage capacity, and bandwidth to handle the expected query load.
2. **Indexing**:
   * **Indexing Process**: Involves parsing documents, extracting relevant data, and storing it in an index. The example provided illustrates how to convert text into an indexable format.
   * **Centralized vs. Distributed Indexing**: The document contrasts a centralized architecture with a distributed one, highlighting the benefits of distributing the indexing process to handle large datasets more efficiently.
3. **Initial Design**:
   * **High-level Design**: Outlines the system architecture, including the API and the details of the indexing and searching processes. The initial design aims to provide a robust framework that meets the outlined requirements.
   * **API Design**: Describes the endpoints and their functionalities, ensuring that the system can interact with other components and users seamlessly.
4. **Final Design**:
   * **Scalability Enhancements**: The initial design is evaluated and improved to enhance scalability. This includes partitioning the index across multiple nodes, using techniques like sharding and replication.
   * **Fault Tolerance**: Incorporates mechanisms to ensure that the system remains operational even when some components fail. This involves redundancy and data replication strategies.
5. **Evaluation**:
   * **Performance Metrics**: The final design is evaluated against the initial requirements. Metrics such as query response time, throughput, and system availability are used to measure performance.
   * **Scalability Tests**: The system is tested under various load conditions to ensure it can handle increased traffic without degradation in performance.
   * **Cost Analysis**: Estimates the operational costs, including infrastructure and maintenance, to ensure the system is cost-effective.

### **Additional Considerations**

The document also covers other important components and building blocks of a distributed search system:

* **Load Balancers**: Distribute incoming queries evenly across multiple servers to ensure no single server is overwhelmed.
* **Distributed Cache**: Stores frequently accessed data closer to the user to reduce query response times.
* **Distributed Messaging Queue**: Ensures reliable communication between different components of the system.
* **Rate Limiter**: Prevents abuse by limiting the number of queries a user can make in a given timeframe.

In conclusion, designing a distributed search system involves a detailed understanding of the system's requirements, effective indexing strategies, robust initial and final designs, and thorough evaluation to ensure performance and scalability. By addressing these aspects, the document provides a solid foundation for engineers and managers to design efficient distributed search systems.

2. Requirements of a Distributed Search System's Design.pdf

PDF

As an expert in system design, summarize the following document in great detail.

ChatGPT

The document "Requirements of a Distributed Search System's Design" from the Grokking Modern System Design Interview for Engineers & Managers course outlines the essential requirements and resource estimations needed for designing a distributed search system. The document is structured into sections that detail functional and non-functional requirements, resource estimations, and the building blocks for such a system.

### **Functional Requirements**

1. **Search**: The system must provide relevant content based on user search queries.

### **Non-Functional Requirements**

1. **Availability**: The system should be highly available to users, ensuring it is operational and accessible at all times.
2. **Scalability**: The system must be capable of scaling to handle increasing amounts of data, meaning it can index large datasets efficiently.
3. **Fast Search on Big Data**: Users should receive search results quickly, regardless of the size of the data being searched.
4. **Reduced Cost**: The overall cost of building and maintaining the search system should be minimized.

### **Resource Estimation**

The document provides detailed calculations for estimating the number of servers, storage, and bandwidth required for a distributed search system using YouTube as an example.

#### **Number of Servers Estimation**

* **Assumptions**:
  + Daily active users using the search feature: 3 million.
  + Requests a single server can handle: 1,000.
* **Calculation**:
  + Total search requests at one time: 3 million.
  + Number of servers needed: 3 million / 1,000 = 3,000 servers.

#### **Storage Estimation**

* **Assumptions**:
  + Size of a single JSON document (video metadata): 200 KB.
  + Number of unique terms per document: 1,000.
  + Storage required per term: 100 Bytes.
* **Calculation**:
  + Total storage per video: 200 KB (metadata) + (1,000 terms \* 100 Bytes) = 300 KB.
  + Average number of videos uploaded per day: 6,000.
  + Total storage required per day: 6,000 \* 300 KB = 1.8 GB.

#### **Bandwidth Estimation**

* **Incoming Traffic**:
  + Assumptions:
    - Number of search requests per day: 150 million.
    - Search query size: 100 Bytes.
  + Calculation:
    - Bandwidth required per second: (150 million / 86,400 seconds) \* 100 Bytes = 1.39 Mbps.
* **Outgoing Traffic**:
  + Assumptions:
    - Number of suggested videos per search query: 80.
    - Size per suggestion: 50 Bytes.
    - Total response size per query: 4,000 Bytes.
  + Calculation:
    - Bandwidth required per second: (150 million / 86,400 seconds) \* 4,000 Bytes = 55.56 Mbps.

### **Building Blocks**

The document mentions the use of distributed storage systems, such as blob stores, to store the data to be indexed and the index itself.

### **Summary**

In summary, the document provides a thorough understanding of the requirements needed to design a distributed search system. It highlights the importance of scalability, availability, and cost efficiency. The detailed resource estimations ensure that the system can handle large volumes of data and user queries efficiently. The use of distributed storage systems as a building block ensures that the system can manage and store vast amounts of data effectively.

3. Indexing in a Distributed Search.pdf

PDF

As an expert in system design, summarize the following document in great detail.

ChatGPT

The document "Indexing in a Distributed Search" from the Grokking Modern System Design Interview for Engineers & Managers course provides an in-depth exploration of the concepts and techniques involved in building and managing indexing in a distributed search system. The document is structured into sections that cover the basics of indexing, the creation of a searchable index, the inverted index, and the issues associated with centralized indexing.

### **Indexing Overview**

Indexing is the process of organizing and manipulating data to facilitate fast and accurate information retrieval. It involves creating data structures that allow quick search and retrieval operations.

### **Building a Searchable Index**

The simplest way to build a searchable index is to assign a unique ID to each document and store it in a database table. For instance:

csharp

ID | Document Content

1 | Elasticsearch is a distributed and analytics engine based on REST APIs.

2 | Elasticsearch is a Lucene library-based search engine.

3 | Elasticsearch is a distributed search and analytics engine built on Apache Lucene.

In a real-world scenario, documents can be much larger, making this approach inefficient for quick searches as it requires traversing all documents and counting occurrences of search strings.

### **Inverted Index**

An inverted index is a more efficient data structure for search operations. It employs a document-term matrix, splitting documents into individual words and creating mappings for each term:

lua

Term | Mapping (Document IDs, Frequencies, Positions)

elasticsearch| ([1, 2, 3], [1, 1, 1], [[1], [1], [1]])

distributed | ([1, 3], [1, 1], [[4], [4]])

search | ([1, 2, 3], [1, 1, 1], [[6], [4], [5]])

This structure allows for faster searches by directly mapping terms to their occurrences in documents.

#### **Advantages of Inverted Index**

* **Full-text Searches**: Facilitates efficient full-text searches.
* **Search Time Reduction**: Reduces the time required to count term occurrences at runtime.

#### **Disadvantages of Inverted Index**

* **Storage Overhead**: Requires additional storage for maintaining the index alongside the actual documents.
* **Maintenance Costs**: Involves processing costs for adding, updating, or deleting documents to keep the index current.

### **Searching from an Inverted Index**

When searching for terms, the system retrieves lists of document IDs, frequencies, and positions. For example, searching for "search engine" retrieves:

css

Term | Mapping

search | ([1, 2, 3], [1, 1, 1], [[6], [4], [5]])

engine | ([1, 2, 3], [1, 1, 1], [[9], [5], [8]])

Both terms are found in documents 1, 2, and 3, and the system uses these mappings to quickly locate relevant documents.

### **Factors of Index Design**

Key considerations in index design include:

* **Index Size**: Memory required to store the index.
* **Search Speed**: Efficiency of finding terms.
* **Maintenance**: Ease of updating the index with new or deleted documents.
* **Fault Tolerance**: System's ability to remain reliable despite failures or corruption.
* **Resilience**: Resistance to manipulation and gaming of search results.

### **Centralized vs. Distributed Indexing**

#### **Centralized Indexing**

In a centralized system, all components run on a single node. This approach has significant drawbacks:

* **Single Point of Failure (SPOF)**: The system fails if the central node goes down.
* **Server Overload**: High query volume can overwhelm the server.
* **Large Index Size**: Managing a large index on a single server increases complexity and cost.

### **Moving to a Distributed System**

Distributed indexing addresses the limitations of centralized systems by spreading the load across multiple nodes. This approach:

* **Reduces SPOF**: Multiple nodes provide redundancy.
* **Handles Larger Indexes**: Distributes the index across several machines, making it more manageable and cost-effective.
* **Improves Search Speed**: Parallel processing across nodes speeds up query responses.

### **Conclusion**

Indexing in a distributed search system is crucial for efficient data retrieval. The document outlines the advantages of inverted indexes and the necessity of moving from centralized to distributed systems to handle scalability, fault tolerance, and maintenance efficiently.

This detailed exploration of indexing mechanisms and their implementation in distributed systems provides a solid foundation for designing and managing efficient search systems.

4. Design of a Distributed Search.pdf

PDF

As an expert in system design, summarize the following document in great detail.

ChatGPT

The document "Design of a Distributed Search" from the Grokking Modern System Design Interview for Engineers & Managers course provides a comprehensive guide on designing a distributed search system that can handle a large number of queries per second. The document is structured into sections covering high-level design, API design, detailed discussions on distributed indexing and searching, replication, and replication factor and replica distribution.

### **High-level Design**

The distributed search system design includes two phases:

1. **Offline Phase**: Involves data crawling and indexing without user interaction.
2. **Online Phase**: Involves searching for results based on user queries.

#### **Components:**

* **Crawler**: Collects content from the intended resources (e.g., YouTube videos) and stores the extracted content in JSON documents within a distributed storage system.
* **Indexer**: Fetches documents from the distributed storage, processes them using a distributed data processing system like MapReduce, and creates an inverted index stored back in the distributed storage.
* **Searcher**: Parses user search queries, retrieves relevant mappings from the index, and returns the most matched results to the user, handling misspellings and ranking results.

### **API Design**

The API design is straightforward due to the simplicity of user interaction, consisting primarily of a search function:

* **search(query)**: Takes a textual query entered by the user and returns search results based on the indexed data.

### **Detailed Discussion**

#### **Distributed Indexing and Searching**

To develop a distributed indexing and searching system, the documents are partitioned and indexed across multiple low-cost nodes connected in a cluster. The two common techniques for data partitioning are:

* **Document Partitioning**: Each node indexes a subset of documents assigned to it.
* **Term Partitioning**: The dictionary of terms is split, with each subset of terms residing at a single node.

In practice, document partitioning is preferred due to its simplicity and lower inter-node communication requirements.

#### **Distributed Design for Index Construction and Querying**

The design uses a cluster of low-cost nodes managed by a cluster manager, which employs the MapReduce programming model for parallel and distributed index computation. The indexing process involves:

* **Partitioning**: The cluster manager splits the document set into partitions.
* **Parallel Indexing**: Each partition is indexed simultaneously across nodes, producing multiple small inverted indexes.
* **Searching**: User queries are run in parallel across these small indexes, and results are merged and sorted before being returned to the user.

### **Replication**

To ensure high availability and fault tolerance, the system uses replication:

* **Replication Factor**: Typically set to three, meaning each partition is replicated across three nodes. One node acts as the primary, while the others are replicas.
* **Replica Distribution**: Replicas are distributed across different availability zones to improve performance and system resilience.

#### **Indexing with Replicas**

Each partition is forwarded to all replicas for index computation, ensuring that if a primary node fails, other replicas can continue indexing without interruption.

#### **Searching with Replicas**

Queries are distributed across multiple replicas, improving system scalability and availability by allowing the system to handle more queries concurrently.

### **Summary**

The design achieves scalability and fault tolerance by:

* **Parallel Indexing and Searching**: Co-located on the same nodes to handle large datasets efficiently.
* **Replication**: Ensures system resilience and high availability, with replicas distributed across different availability zones.

### **Conclusion**

The proposed design effectively handles large volumes of data and high query rates, offering scalability, low latency, and robustness through parallel processing and replication. Future lessons will address potential drawbacks and further optimizations for the distributed search system.

5. Scaling Search and Indexing.pdf

PDF

As an expert in system design, summarize the following document in great detail.

ChatGPT

The document "Scaling Search and Indexing" from the Grokking Modern System Design Interview for Engineers & Managers course provides a detailed analysis of scaling strategies for search and indexing systems in a distributed environment. Here is a comprehensive summary of the document:

### **Key Issues in the Existing Design**

1. **Colocated Indexing and Searching**:
   * Indexing and searching are performed on the same nodes, leading to resource contention.
   * Both operations are resource-intensive and negatively impact each other's performance.
   * This design does not scale efficiently with varying indexing and search loads, causing potential performance bottlenecks.
2. **Index Recomputation**:
   * Each replica node computes the index individually, which is resource-inefficient.
   * Index computation involves multiple stages and is highly resource-intensive.
   * The recommended approach is to compute the index once on the primary node and replicate it across other nodes.

### **Proposed Solution**

The document suggests separating the indexing and searching processes to avoid the drawbacks of the colocated design. The advancements in networking, virtualization, and cloud computing enable this separation efficiently.

### **Components of the Redesigned System**

1. **Indexer**:
   * A group of nodes dedicated to computing the index.
   * Uses a MapReduce model for distributed indexing.
2. **Distributed Storage**:
   * Stores the partitions and computed index.
   * Ensures data is accessible across different nodes.
3. **Searcher**:
   * Nodes dedicated to performing search queries.
   * Fetches and caches index files for quick access.

### **Detailed Design**

#### **Indexing Process**

* **MapReduce Framework**:
  + Utilizes a cluster manager and worker nodes (Mappers and Reducers).
  + **Map Phase**: Mappers extract and filter terms from document partitions and generate inverted indexes.
  + **Reduce Phase**: Reducers combine mappings from Mappers to generate a summarized index.
  + The output is an aggregated inverted index stored in distributed storage.
* **Process Flow**:
  + Document set partitioning.
  + Parallel indexing using low-cost nodes in a cluster.
  + Index storage on local and distributed storage for redundancy and fault tolerance.

#### **Searching Process**

* **Search Query Handling**:
  + Searcher nodes download index files and maintain a cache of frequently queried terms.
  + User queries are distributed across searcher nodes, which generate responses based on their respective indices.
  + A merger node combines the search results from different nodes and presents them to the user.
* **Optimization**:
  + Frequent queries are served from RAM to reduce latency.
  + The system is designed to scale horizontally by adding more searcher nodes as query load increases.

### **Benefits of the Redesigned System**

* **Resource Efficiency**:
  + Avoids redundant index computation by centralizing the indexing process.
  + Reduces CPU and memory usage on replica nodes.
* **Scalability**:
  + Separating indexing from searching allows each process to scale independently.
  + Utilizes cloud technologies for scalable storage and bandwidth.
* **Fault Tolerance**:
  + Index and search operations are distributed across multiple nodes, reducing the impact of node failures.
  + Redundant storage of index files ensures data availability.

### **Summary**

The redesign addresses the scalability and resource efficiency issues of the initial colocated design. By separating indexing and searching, and leveraging distributed storage and the MapReduce framework, the system can handle large volumes of data and queries efficiently. The document highlights the importance of parallel processing and resource optimization in building a scalable and performant distributed search system.

This approach ensures that the search system remains responsive and reliable, even as the data and query loads grow, making it well-suited for large-scale applications like web search engines and content delivery networks.

6. Evaluation of a Distributed Search's Design.pdf

PDF

As an expert in system design, summarize the following document in great detail.

ChatGPT

The document "Evaluation of a Distributed Search's Design" from the Grokking Modern System Design Interview for Engineers & Managers course provides an in-depth analysis of how the proposed distributed search system meets key requirements such as availability, scalability, speed on large datasets, and cost efficiency. Here's a detailed summary:

### **Availability**

* **Distributed Storage**: Documents crawled by the indexer and inverted indexes generated by indexing nodes are stored in distributed storage.
* **Data Replication**: Data is replicated across multiple regions in distributed storage, facilitating cross-region deployment for both indexing and search.
* **Deployment Strategy**: Clusters of indexing and search nodes are deployed in different availability zones. If one zone fails, another can handle the requests, ensuring high availability.
* **Node Failure Handling**: Within each cluster, if a node fails, another node can take its place. This redundancy helps maintain availability.
* **Offline Indexing**: Indexing operations are performed offline and do not need to be replicated synchronously. This means that user search queries do not have to wait for the latest data to be indexed and replicated.

### **Scalability**

* **Partitioning**: The system uses partitioning to scale. Increasing the number of partitions and adding more nodes to the indexing and search clusters allows for scalable data indexing and querying.
* **Process Isolation**: Strong isolation of indexing and search processes helps them scale independently and dynamically, accommodating varying loads.

### **Fast Search on Big Data**

* **Parallel Search Queries**: The system utilizes multiple nodes, each performing search queries in parallel on smaller inverted indexes. The results from each node are merged and returned to the user, ensuring fast query responses even on large datasets.

### **Reduced Cost**

* **Low-cost Machines**: The design uses cheaper machines for computing indexes and performing searches, reducing overall costs.
* **Efficient Index Computation**: If a node fails, the system does not need to recompute the complete index. Only some documents need to be re-indexed, further minimizing resource usage and cost.

### **Conclusion**

The document concludes that a distributed search system is essential for almost every application, as it is impractical to develop a search system that runs on a single node due to scalability and resource constraints. The proposed design uses a parallel computation framework and low-cost machines to build a search system that is available, scalable, and highly performant.

### **Key Points:**

1. **Availability**: Achieved through data replication, cross-region deployment, and redundancy within clusters.
2. **Scalability**: Facilitated by partitioning data and isolating indexing and search processes.
3. **Performance**: Ensured by parallel processing of search queries across multiple nodes.
4. **Cost Efficiency**: Maintained by using low-cost machines and minimizing the need for full index recomputation.

This evaluation highlights the strengths of the distributed search design in meeting the essential requirements of availability, scalability, performance, and cost efficiency, making it a robust solution for handling large-scale search operations.

ChatGPT can make mistakes. Check important info.

ChatGPT