

Searching and Indexing of Big Data

Full text Indexing and Searching

Indexing with Lucene

Distributed Searching with elastic search

Full text indexing and searching

- Full text searching and indexing refers to the process of efficiently searching large amounts of text data for specific keywords or phrases.
- Indexing is the process of creating an index of the text data, mapping keywords to the location of their corresponding data in the text. This allows for faster searches, as the index can be used to quickly identify the relevant documents, rather than searching through every document in the dataset.
- Full text search engines, such as Elasticsearch, Apache Solr, or Algolia, use various algorithms, such as inverted indices, to perform the indexing and searching processes.



● Indexing

- Indexing is the initial part of all search applications.
- Its goal is to process the original data into a highly efficient cross-reference lookup in order to facilitate rapid searching.
- Indexing is a way to optimize the performance of a database by minimizing the number of disk accesses required when a query is processed.
- It is a data structure technique which is used to quickly locate and access the data in a database.

● Searching

- Searching is the process of looking up the words in an index to find documents where they appear

Indexing Process

- Acquiring the content
- Build documents
- Document analysis
- Indexing the document

Indexing Process

- **Acquiring the content :** Acquiring the content in index processing refers to the first step in preparing data for efficient search and retrieval in big data systems. This involves obtaining the data that needs to be indexed, which could be from various sources like databases, text files, web pages, etc. The data must be collected and consolidated into a format that can be easily processed and indexed. This step is critical as the quality and accuracy of the indexing process depends heavily on the quality of the data that is being indexed.
- **Build documents :** Building documents in index processing refers to the second step in preparing data for efficient search and retrieval in big data systems. In this step, the raw data that was obtained in the "Acquiring the Content" step is converted into a structured format, such as HTML, XML, or JSON, to enable easy indexing. The structured format provides a clear and consistent representation of the data, which makes it easier to analyze and index. This step is crucial in ensuring that the data is organized and ready for the next step, "Document Analysis."

Indexing Process

- **Document analysis:** The pre-processed documents are analyzed to extract meaningful information such as keywords, phrases, and entities. This information is used to enhance the search capabilities of the index.
- **Tokenization:** The analyzed documents are then tokenized, which means that they are divided into smaller units, such as words or phrases. This allows for more fine-grained searching and indexing.
- **Normalization:** Tokens are normalized to ensure consistency in the representation of words and phrases. This may include Case normalization (i.e converting them to lowercase), removing punctuation, Stemming and lemmatization, Stop word removal, Numbers normalization, Named Entity Recognition .
- **Filtering:** Stop words and other irrelevant tokens are filtered out to reduce the size of the index and to improve the quality of the search results.


Indexing Process

- **Indexing the document** : The final step is to index the documents, which means creating a mapping between the tokens and the documents in which they appear. The resulting index allows for fast keyword searches by looking up the corresponding documents in the index, rather than having to search through the entire text corpus.
- These steps form the process of index processing, which is a crucial step in full-text search and information retrieval. The quality of the index and the search results depend on the effectiveness of the index processing steps.

Document Search

- Document search in big data refers to the process of finding specific information within large sets of unstructured or semi-structured data.
- This can be a challenge as traditional search algorithms may not scale well when faced with vast amounts of data. In such cases, advanced technologies like machine learning, natural language processing, and information retrieval techniques are often employed to improve the accuracy and speed of document search in big data environments.
- The goal is to provide relevant results from a large collection of data, in a quick and efficient manner, making it easier for users to find the information they need.

- **Some techniques for document search in big data include:**
 - Text indexing and retrieval: Indexing text data and utilizing algorithms such as inverted indices and vector space models to retrieve relevant documents based on keywords or phrases.
 - Machine learning: Using algorithms such as neural networks and decision trees to understand the context and meaning of text data, and rank documents based on relevancy.
 - Natural language processing (NLP): Applying NLP techniques such as named entity recognition, sentiment analysis, and topic modeling to understand the content of documents and make them searchable.

- 
- **Some techniques for document search in big data include:**
 - Distributed processing: Using distributed systems like Hadoop and Spark to process and store large amounts of data, enabling fast and efficient document search.
 - Faceted search: Allowing users to filter search results based on various attributes, such as date, author, or file type, to quickly locate relevant documents.
 - Metadata management: Storing and utilizing metadata, such as author, date, and file type, to aid in the search and retrieval of documents.

Lucene

- Apache Lucene is a high-performance, open-source full-text search library written in Java. It provides indexing and searching capabilities for a wide range of applications, including document retrieval, enterprise search, and e-commerce search.
- Lucene is designed to be scalable, fast, and flexible, and it can be used to index and search large collections of text-based data, including web pages, emails, and other types of documents. It uses an inverted index, a data structure that maps terms to the documents that contain them, to allow for fast and efficient searching.

Lucene

- Lucene also provides a rich set of features for text analysis, including tokenization, stemming, and stop word removal, which can be used to process and analyze text data. It also provides support for ranking and scoring of search results based on relevance, making it easier to sort and present the most relevant results to users.
- Lucene is widely used in a variety of applications, and it is the foundation of several popular search engines, including Apache Solr and Elastic search. It is also a popular choice for building custom search solutions due to its ease of use, performance, and versatility.

Full-text search using Lucene requires two steps:

- **Indexing:** The first step is to index the text data, which involves preprocessing the text data and creating an inverted index that maps terms to the documents that contain them. During indexing, Lucene performs tokenization, stemming, and stop word removal, as well as any other desired text analysis operations, to simplify and standardize the text data.
- **Searching:** The second step is to search the indexed data, which involves querying the inverted index to find the documents that match the search terms. Lucene provides a rich set of query options, including exact matches, wildcard searches, fuzzy searches, and Boolean searches, to allow for flexible and efficient searching. The results of the search are then ranked and scored based on relevance, and the most relevant results can be presented to the user.

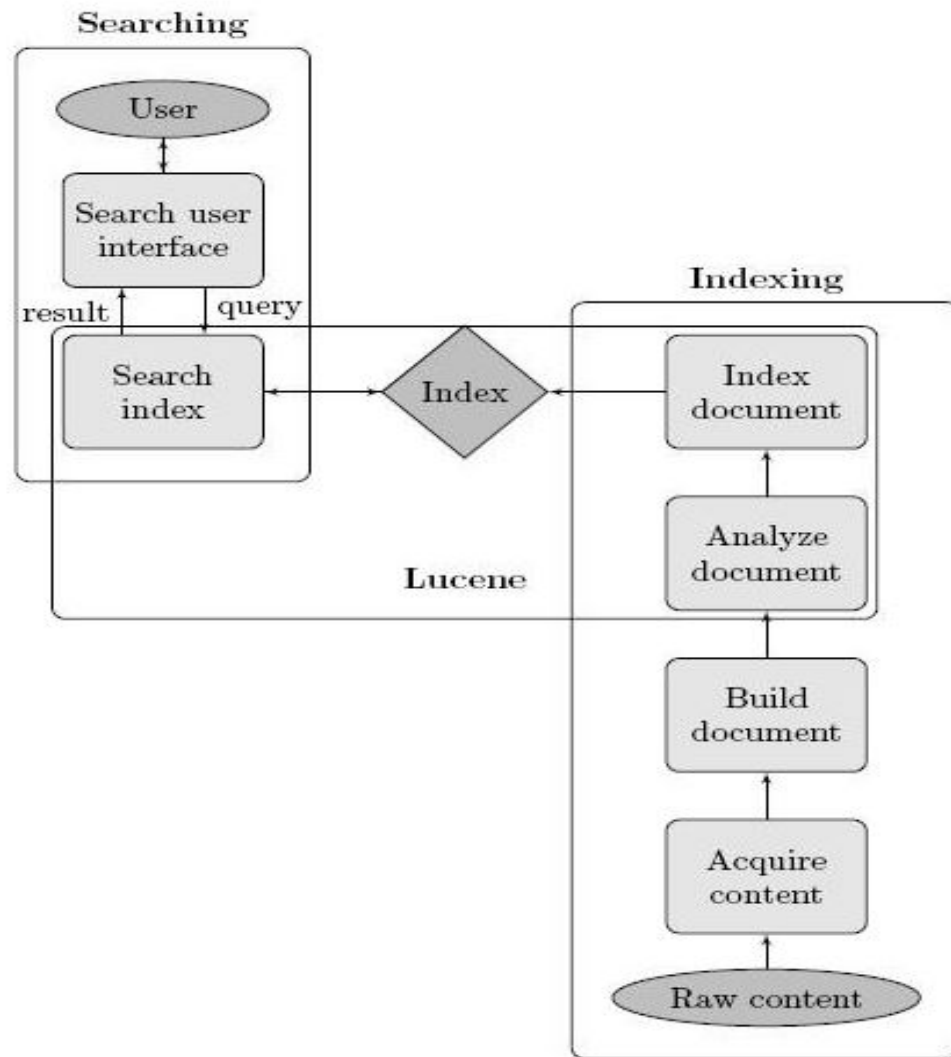



Figure : Typical components of search application architecture with Lucene components highlighted

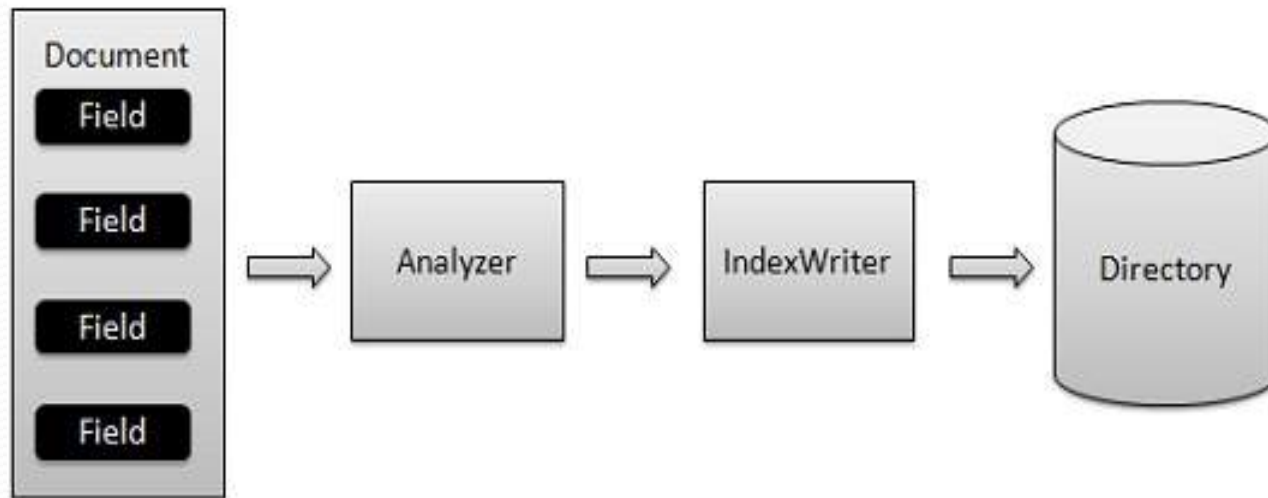
- 
- **Acquire content:** The first step of any search and index application is to collect the target contents on which search application is to be conducted.
 - **Build the document :** The next step is to build the document(s) from the raw content, which the search application can understand and interpret easily.
 - **Analyze the document :** Before the indexing process starts, the document is to be analyzed as to which part of the text is a candidate to be indexed. This process is where the document is analyzed

- **Indexing the document** : Once documents are built and analyzed, the next step is to index them so that this document can be retrieved based on certain keys instead of the entire content of the document. Indexing process is similar to indexes at the end of a book where common words are shown with their page numbers so that these words can be tracked quickly instead of searching the complete book.
- **User interface for search** : Once a database of indexes is ready then the application can make any search. To facilitate a user to make a search, the application must provide a user a mean or a user interface where a user can enter text and start the search process.

- **Build Query** : Once a user makes a request to search a text, the application should prepare a Query object using that text which can be used to inquire index database to get the relevant details.
- **Search Query** : Using a query object, the index database is then checked to get the relevant details and the content documents.
- **Render Results** : Once the result is received, the application should decide on how to show the results to the user using User Interface. How much information is to be shown at first look and so on.

Lucene Indexing Process

- 1. Document Analyzer
- 2. Index Writer
- 3. Index Store



Indexing Process

Analyzer in Lucene

- An Analyzer in Apache Lucene is a component that performs text analysis on the input data and transforms it into a set of terms that can be indexed and searched efficiently. The Analyzer performs several tasks, including:
 - Tokenization: The Analyzer breaks the input text into smaller units called tokens, which are usually individual words or phrases.
 - Lowercasing: The Analyzer typically converts all the tokens to lowercase to make the index case-insensitive.
 - Stop Word Removal: The Analyzer removes common words (stop words) from the tokens, as these words are typically not useful for searching and can slow down the indexing process.
 - Stemming: The Analyzer may perform stemming, which is the process of reducing words to their root form, to standardize the terms and reduce the size of the index.
 - Filtering: The Analyzer may perform additional text processing, such as removing punctuation marks or applying synonym filters, to further simplify and standardize the terms.

Analyzer in Lucene

- In Lucene, different Analyzers can be used depending on the type of data being indexed and the requirements of the search application. Lucene provides several built-in Analyzers, including the Standard Analyzer, the Simple Analyzer, and the Whitespace Analyzer, and custom Analyzers can also be created to meet specific requirements.
- The choice of Analyzer has a significant impact on the performance and accuracy of the search results, as well as the size and complexity of the index. The Analyzer should be chosen carefully to ensure that the text analysis is appropriate for the data and the search requirements.

Index Writer

- The Index Writer in Apache Lucene is a component that is responsible for adding, updating, and deleting documents in the Lucene index. The Index Writer performs several key tasks, including:
 - Document Indexing: The Index Writer is responsible for taking the Document objects that have been created by the text analysis process and adding them to the index. The Index Writer creates an inverted index that maps terms to the documents that contain them, and it stores the index data in an efficient and compact format for fast searching.
 - Document Updating: The Index Writer also provides the ability to update or delete existing documents in the index, which is useful for maintaining the index as the underlying text data changes.

Index Writer

- **Optimizing the Index:** The Index Writer can optimize the index periodically to improve the performance of the search engine. This optimization process involves compressing the index data and organizing it for fast and efficient searching.
- **Transaction Management:** The Index Writer provides transaction management capabilities, which allow for atomic additions, updates, and deletions of documents in the index. This helps to ensure that the index is always in a consistent state, even if the process of indexing is interrupted or fails.
- **The Index Writer is a key component of the Lucene indexing process, and it is used by the Lucene indexer to build and maintain the Lucene index. The Index Writer provides a high-level interface for indexing documents, and it abstracts away the low-level details of the indexing process, making it easier to use and integrate into a larger search engine or application.**

Index Store

- The Index Store in Apache Lucene is the component that is responsible for storing the actual index data on disk or in memory. The Index Store is responsible for:
 - Storing the Inverted Index: The inverted index is a data structure that maps terms to the documents that contain them, and it is the backbone of the Lucene search engine. The Index Store stores the inverted index on disk or in memory, in a format that is optimized for fast searching.
 - Storing the Document Fields: The Index Store also stores the document fields, which are the individual pieces of information that describe each document in the index. The document fields can include information such as the document title, author, and content, and they are used by the Lucene search engine to return relevant search results.

Index Store

- Storing Metadata: The Index Store stores additional metadata about the index, such as the number of documents in the index, the number of terms in the index, and the mapping of terms to document IDs.
- Persistence: The Index Store is responsible for persistently storing the index data on disk, so that it can be retrieved and used later by the Lucene search engine.
- The Index Store is an essential component of the Lucene search engine, as it provides the foundation for fast and efficient searching by storing the inverted index and document fields in a format that is optimized for searching. The choice of Index Store implementation can have a significant impact on the performance and scalability of the Lucene search engine, and different implementations are available for different use cases and performance requirements.

Types of Analyzer

- Apache Lucene provides several types of analyzers that can be used to process text and build the Lucene index. The most common types of analyzers in Lucene are:
 - Standard Analyzer: The Standard Analyzer is the most basic analyzer in Lucene and it is used for simple text processing tasks. The Standard Analyzer splits text into tokens using whitespace and punctuation, lowercases the tokens, and removes stop words (common words like "the", "and", "a", etc.).
 - Simple Analyzer: The Simple Analyzer splits text into tokens using whitespace and lowercases the tokens, but does not remove stop words or perform any other text processing.

Types of Analyzer

- **Whitespace Analyzer:** The Whitespace Analyzer splits text into tokens using whitespace only, and does not perform any other text processing.
- **Stop Analyzer:** The Stop Analyzer splits text into tokens using whitespace and punctuation, and removes stop words, but does not perform any other text processing.

Elastic Search

- Elastic search is an open-source, distributed search and analytics engine designed for handling large amounts of data. I
- t is built on top of Apache Lucene and provides a powerful and flexible search engine for applications that require full-text search, faceted search, geospatial search, and real-time search.
- Some of the key features of Elasticsearch include:
 - Scalability: Elasticsearch is designed to scale horizontally, allowing it to handle increasing amounts of data and search traffic.
 - Distributed Architecture: Elasticsearch is built on a distributed architecture, which allows it to distribute data and processing across multiple nodes, making it highly available and fault-tolerant.

Elastic Search

- RESTful API: Elastic search provides a RESTful API that makes it easy to interact with the search engine and manage data.
- Real-time Search: Elastic search provides real-time search, allowing applications to search and retrieve results as soon as they are indexed.
- Multi-Language Support: Elastic search supports multiple languages, including English, Chinese, French, German, and many others.
- Plugins and Integrations: Elastic search provides a large number of plugins and integrations, including plugins for security, monitoring, and alerting, as well as integrations with other technologies, such as Kibana, Logstash, and Beats.
- Elastic search is widely used for a variety of use cases, including full-text search, log analytics, business intelligence, and more. It is fast, scalable, and highly flexible, making it a popular choice for developers and organizations who need a powerful search engine for their applications.

Key Concepts of elastic search

- Elastic search is a powerful and flexible search and analytics engine that is built on top of Apache Lucene. There are several key concepts that are essential to understand when working with Elastic search:
 - Index: An index in Elastic search is a collection of documents that have similar characteristics. An index can be thought of as a database table in a relational database, and a document as a row in the table.
 - Document: A document in Elastic search is a basic unit of data that is stored in an index. A document can be thought of as a JSON object, and can contain any number of fields, each with its own value.

Key Concepts of elastic search

- Node: A node in Elasticsearch is a single instance of the Elastic search software. Nodes can be combined to form a cluster, which allows for increased performance, scalability, and fault tolerance.
- Cluster: A cluster in Elastic search is a collection of nodes that work together to store and index data. Clusters provide a way to distribute data and processing across multiple nodes, making them highly available and fault-tolerant.
- Shards: A shard in Elastic search is a single index that is divided into smaller pieces and stored on different nodes in a cluster. This allows data to be distributed across multiple nodes, improving performance, scalability, and fault tolerance.

Key Concepts of elastic search

- Replicas: A replica in Elastic search is a copy of a shard that is stored on another node in the same cluster. Replicas provide increased availability, as well as the ability to serve search requests in the event that a primary shard is unavailable.
- Mapping: A mapping in Elastic search defines the structure of an index and the types of fields that it contains. Mappings are used to specify the data types of fields, as well as to control how the data is indexed and stored.
- Analyzer: An analyzer in Elastic search is a component that is used to tokenize text and convert it into a form that can be easily searched and indexed. Analyzers play an important role in the search process, as they control how text is transformed into a searchable form.

Advantages

- Elastic search is a popular open-source search and analytics engine that has several advantages over traditional search engines:
 - **Scalability:** Elastic search is designed to scale horizontally, allowing it to handle large amounts of data and traffic. This makes it an ideal choice for applications with high traffic or large data sets.
 - **Real-time search:** Elastic search provides real-time search results, meaning that as soon as new data is added, it is immediately searchable. This makes it an ideal choice for applications where up-to-date information is critical.

Advantages

- **Distributed architecture:** Elastic search has a distributed architecture, meaning that data is distributed across multiple nodes in a cluster. This improves performance, scalability, and fault tolerance, as well as allowing for fast, parallel processing of search requests.
- **Flexibility:** Elastic search supports a wide range of use cases, from log analysis and monitoring to full-text search and analytics. Its flexibility makes it an ideal choice for a wide range of applications and use cases.
- **Analytics:** Elastic search includes built-in support for analytics, allowing for powerful data analysis and reporting. This makes it an ideal choice for applications that require complex data analysis and reporting.

Advantages

- **Multi-language support:** Elastic search supports multiple languages, including English, Chinese, French, German, and many others. This makes it an ideal choice for international applications and users.
- **Large community:** Elastic search has a large and active community of users, developers, and contributors. This community provides a wealth of resources, including documentation, tutorials, and support, making it easy for developers to get started and to find help when needed.

Reference

- <https://medium.com/couchbase/full-text-search-indexing-best-practices-by-use-case-606dd620349e>
- <http://atlassearchrestaurants.com>
- <https://medium.com/expedia-group-tech/getting-started-with-elastic-search-6af62d7df8dd>
- <https://medium.com/velotio-perspectives/elasticsearch-101-fundamentals-core-components-a1fdc6090a5e>
- <https://medium.com/@igorkopanev/a-brief-explanation-of-the-inverted-index-f082993f8605>

inverted index

- An inverted index is a data structure used in full-text search and information retrieval to efficiently search for keywords or phrases in a large corpus of text.
- It maps words or terms to the documents in which they appear, rather than mapping documents to their terms. This allows for fast searching of keywords by looking up the corresponding documents in the index, instead of having to search through the entire text corpus.
- The inverted index stores a list of the documents that contain each word or term, along with their frequency and position within the documents. This information is used to rank search results based on relevance. Inverted indices are commonly used in search engines and database systems that need to perform fast text searches.