5. Searching and Indexing

Lucene

It is an open source Java-based serach library providing APIs for performing common search and serach related tasks like indexing, querying, highlighting, language analysis and many others. It is adopted in popular websites/applications like Twitter, Netflix and Instagram and many other open search based apps. Main capabilities of Lucene are centered on the creation, maintenance and accessibility of Lucene inverted index.

Features

Lucene features can be broken down into four categories

1. Language Analysis: The analysis capabilities are responsible for taking in content in the form of documents to be indexed or queries to be searched and converting them to appropriate internal representation that can be used as needed.
2. Indexing and Storage: Indexing of user defined documents, Storage of user deifned documents, *lock-free indexing*, Near real time indexing, segmented indexing, abstractions, transactional support.
3. Querying: A variety of query options along with the ability to filter, page and sort results as well as perform pseudo relevance feedback. Lucene provides over 50 types of query representations, several query parsers and a query parsing framework.
4. Ancillary Features: Ancillary modules contain a variety of capabilities commonly used in building serach-based applications.

Architecture and Implementation

Foundations

1. Text Analysis: Text analysis chain produces a stream of tokens from the input data in field.
2. Finite State Automata: The in memory portion of the inverted index is implemented with finite state transduces(FST) package.

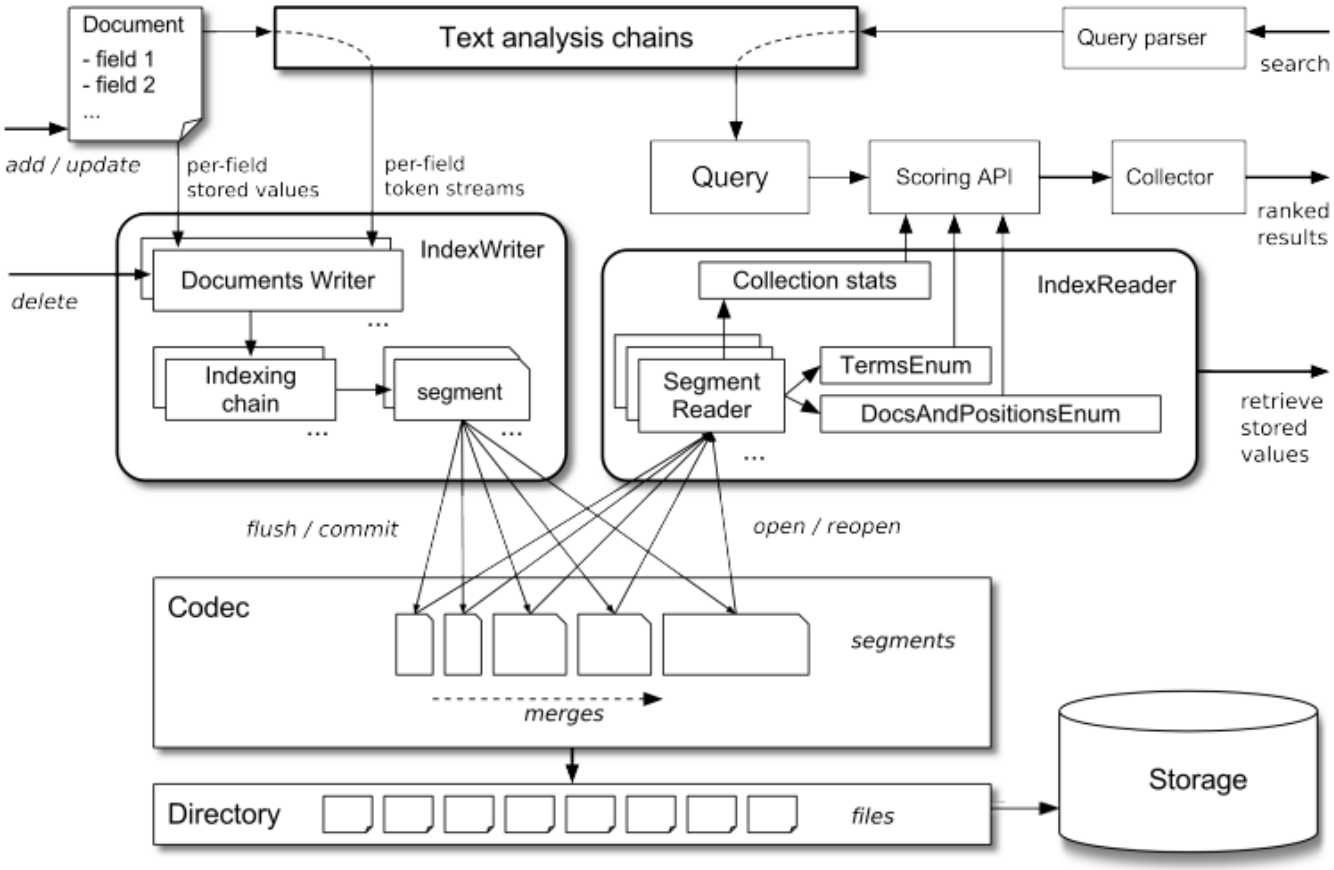


Figure 1 Lucene's Architecture

Indexing

Lucene uses well-known inverted index representation with additional functionality of keeping adjacent non-inverted data on a per-document basis. Incremental updates are supported and stored in index extents(called *segments*) that are periodically merged into larger segments to minimize total number of index parts.

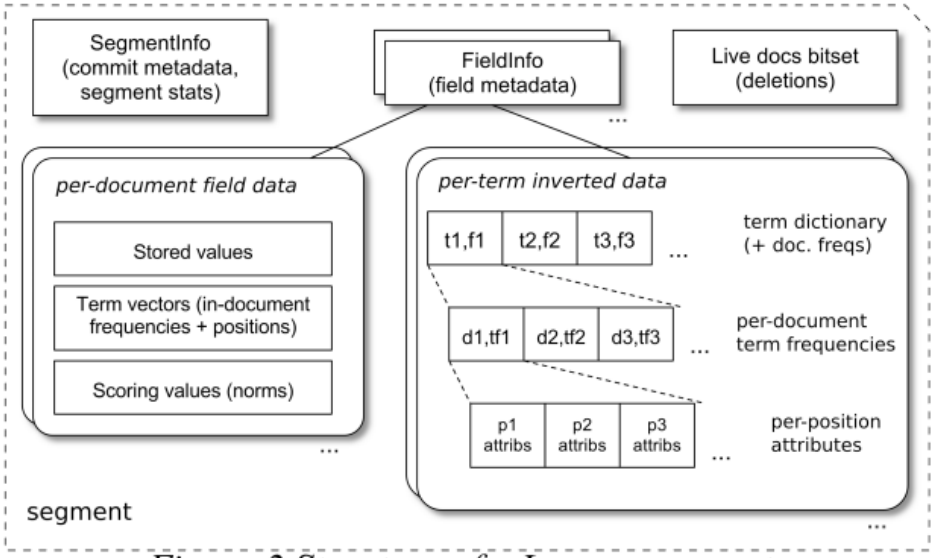


Figure 2 Structure of a Lucene segment.

1. Document Model: Documents are modeled ass flat ordered list of fields with content. Fields have *name, content data, float weight(used for scoring)* and other attributes depending on their type. Docuements are not required to have unique identifier.

2. Field Types: There are mainly two broad categories of fields in Lucene documents - those that carry content to be inverted(indexed fields) and those with content to be stored as-in(stored fields). Indexed fields can be provided in plain text, which will be passed through text analysis pipeline, or in its final form of sequence of tokens with attributes(called *token stream*). Token streams are then inverted and added to in-memory segmens which are periodically flushed and merged. Stored fields are typically used for storing auxiliary per-document data that is not searchable but would be cumbersome to obtain otherwise.

3. Indexing Chain: The resulting token stream is finally processed by the indexing chain and the suppoted attributes are added to the respective posting lists for each term.

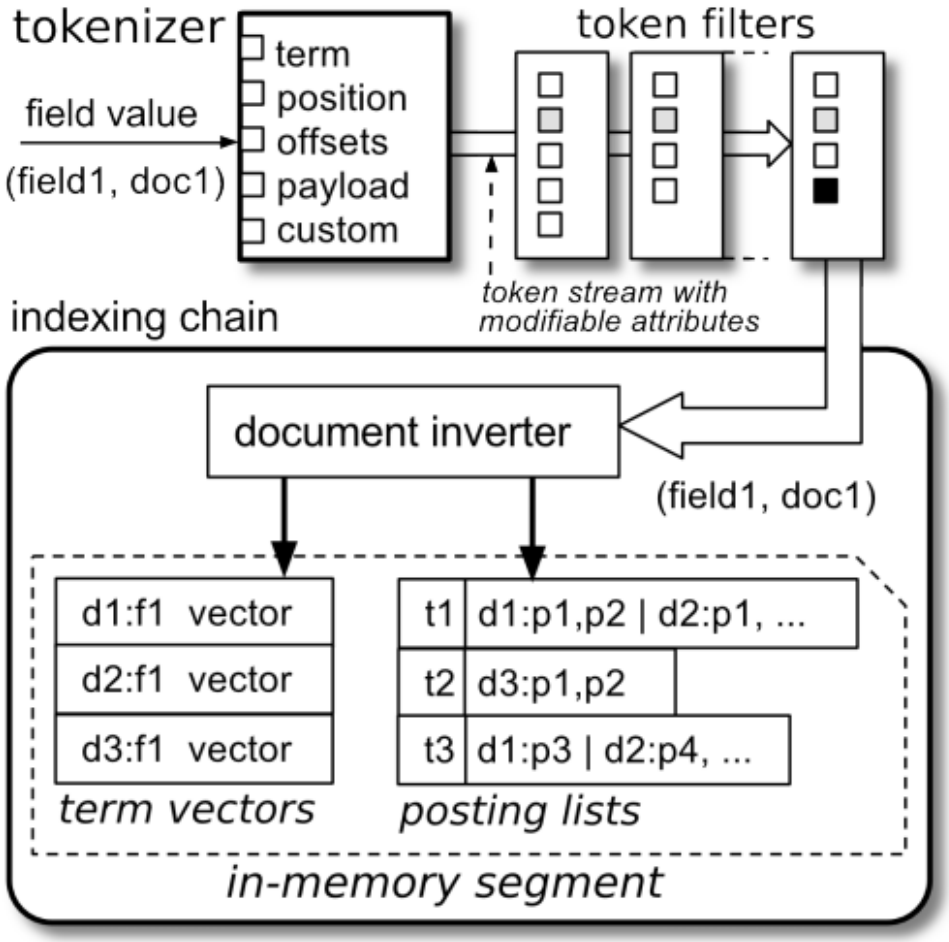


Figure 3 Indexing Process

Searching

Lucene's searching concerns can be broken down into a few key areas.

1. Query Model and Types

   Lucene doesn't enforce a particular query language, instead it uses Query objects to perform searches. Several Queries are provided as building locks to express queries and developers can construct their own programmatically or via a Query Parser. Query types in Lucene 4.0 include: *terms queries, boolean queries, proximity queries, position based queries, disjunction-max query, payload query.* Typically a search is parsed by a Query Parser into a Query Tree.

2. Query Evaluation

   When a query is executed, each inverted index segment is processed sequentially for efficiency. For each index segment, the Query generates a Scorer. *Scorers* typically store documents with a document-at-a-time(DAAT) strategy. Someties term-at-a-time like strategy is also used when number of terms is low. Scorers at leaf nodes in Query Tree

typically compute the score by using raw index statistics whereas the Scorers higher up operate on sub-scorers. Finally a *Collector* is responsible for actually consuming these Scorers and doing something with the results. For example: populating a priority queue with the top-N documents.

3. Similarity

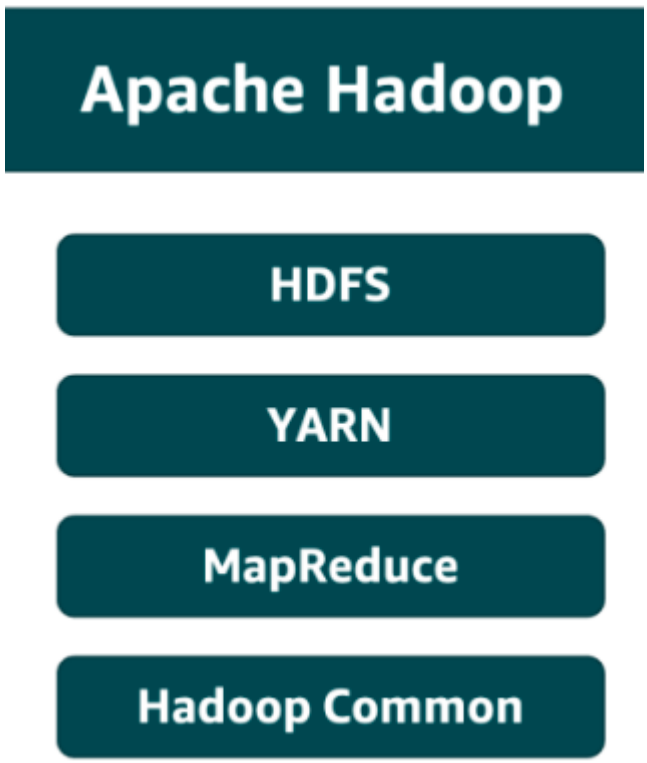
   The similarity class implements a policy for scoring terms ans query clauses taking into account term and global index statistics as well as specifics of a query.

4. Common Search Extensions

   Extended query processing capabilities to support(on top fo basic keyword search) to support easier navigation of search results.

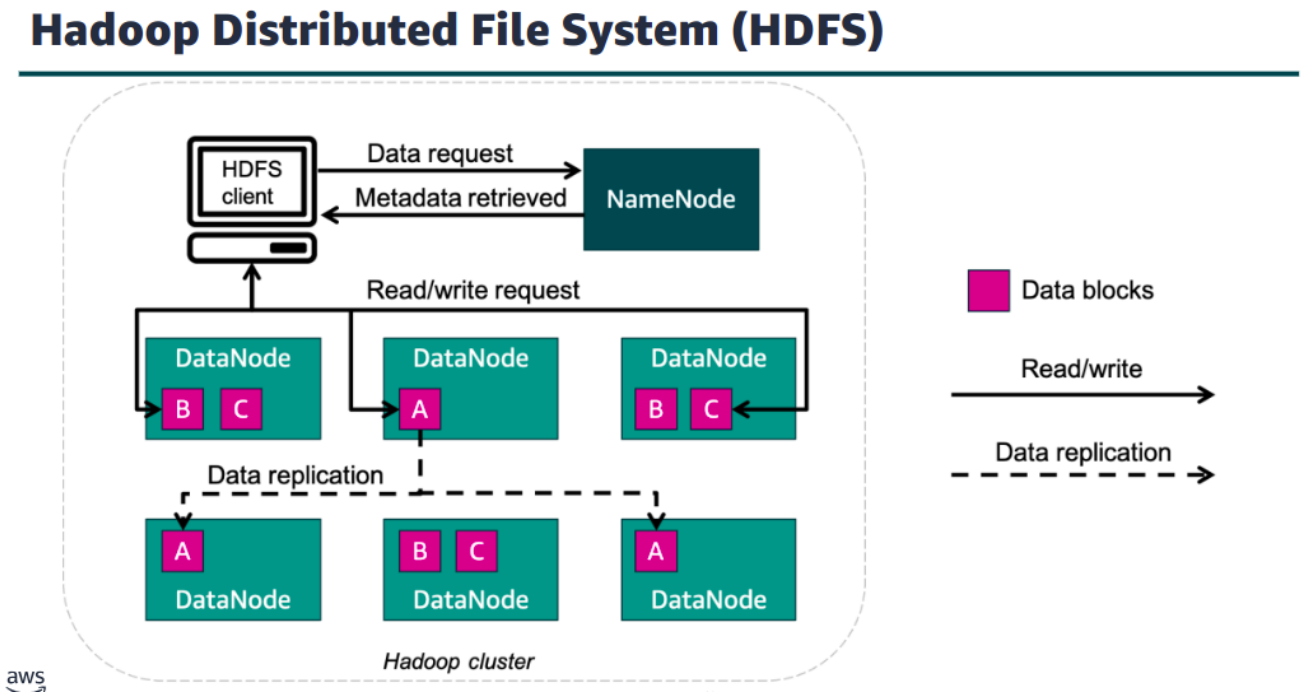
6. Hadoop

It is an open-source distributed computing framework for processing and storing large datasets in a distributed manner across clusters of commodity hardware. It maps tasks to nodes within a clusters of servers.

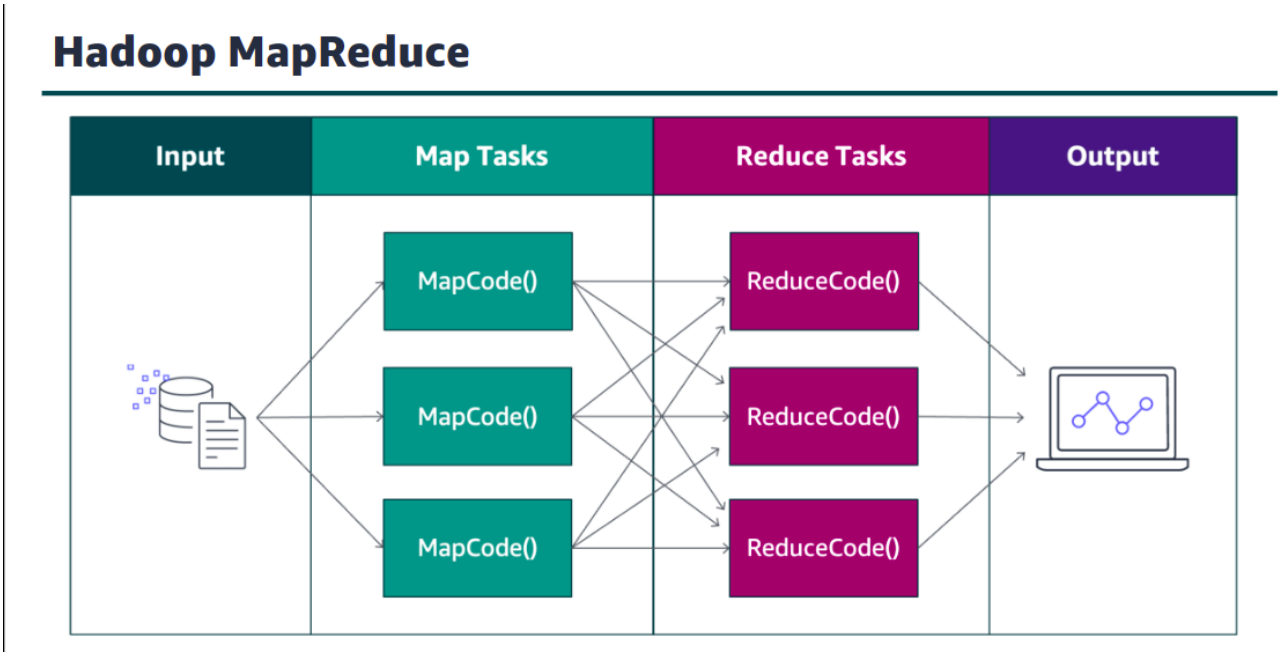


Components of Hadoop

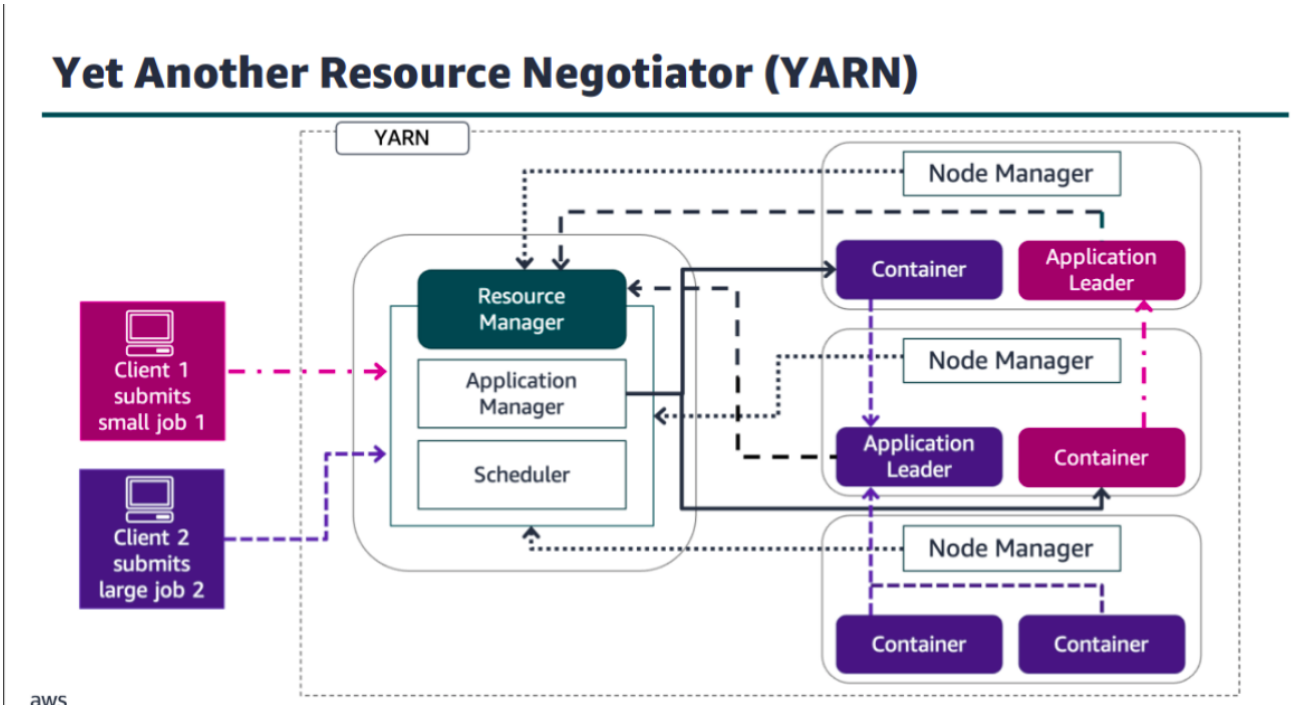
1. HDFS: HDFS is a distributed file system designed to store large volumes of data reliably across multiple machines.



2. MapReduce: MapReduce is a programming model and processing engine for parallel processing of large datasets across distributed clusters.



3. YARN(Yet Another Resoource Negotiator): YARN is a resource management and job scheduling framework in Hadoop.



- *Resource Manager*: Controls the use of resources within hadoop cluster. Contains *Scheduler* responsible for allocating resources to the applications based on their requirements and *Application Manager* which accepts job submissions and provides restart on failure.
- *Node Manager*: Controls use of resources within a node.
- *Application Leader*: Works with Resource Manager and Node Manager to acquire resources.
- *Containers*: Collection of cluster resources.

4. Hadoop Common: Hadoop Common provides the libraries, utilities, and APIs used by other Hadoop components.