Search Engine Architecture

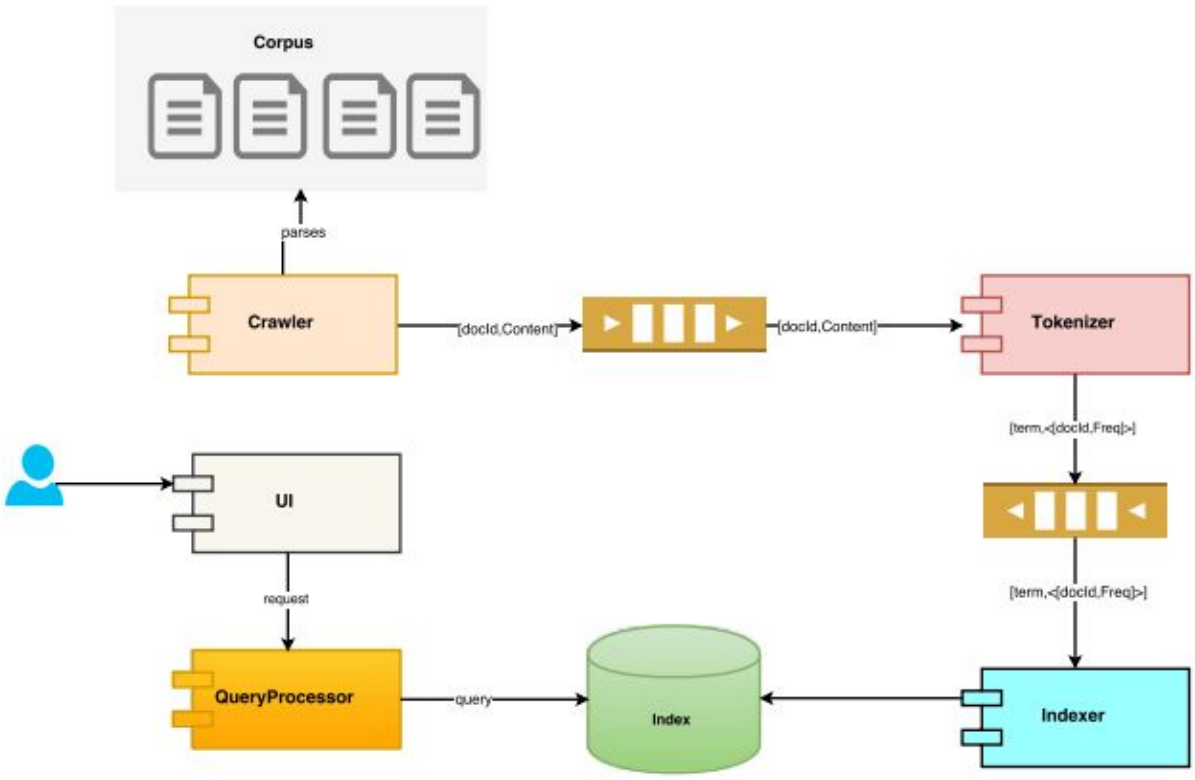
Maysam Mokarian, Spring 2018

Introduction

This document is to provide the architecture design for a search engine. The documents in the corpus is parsed and indexed in this architecture. Additionally The user query will be search in the index and the relevant documents will be served to the user.

High level architecture design

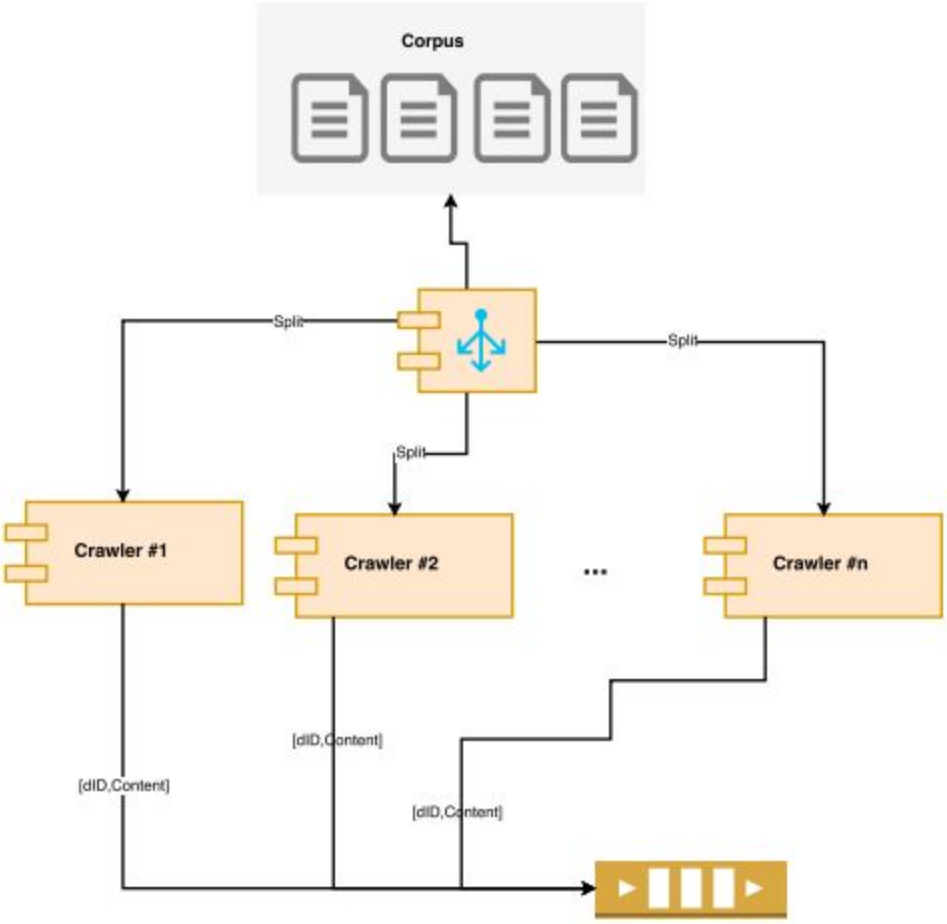
Below is the high level architecture of a search engine:



figure(1) High level architecture of a search engine

Crawler

Below is a more detailed architecture of the Crawler component of a search engine:



figure(2) High level architecture of Crawler

Tasks executed by a Crawler machine:

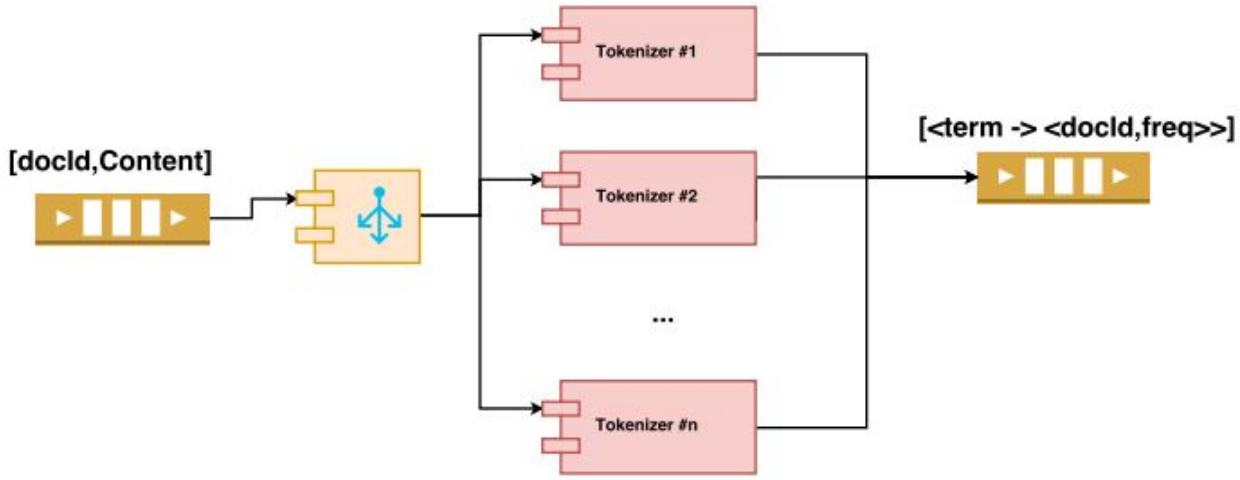
Title: Parse documents in the corpus
Performed tasks:

- Keep track of parsed documents

Output:[docID, docContent]

Tokenizer

Below is a more detailed architecture of the Tokenizer component of a search engine:



figure(3) High level architecture of Tokenizer

Tasks executed by a Tokenizer machine:

Title: Tokenizes the received documents and creates inverted index for each split
Performed tasks:

- Strip HTML/XML tags
- Stemming and lemmatization (employee, employer, employment -> employ)
- Remove the stopwords ("a", "and", "but", "how", "or", and "what)
- Tokenization
- Sorting

Output: [<term -> <docId,freq>>]

Input example for tokenizer:

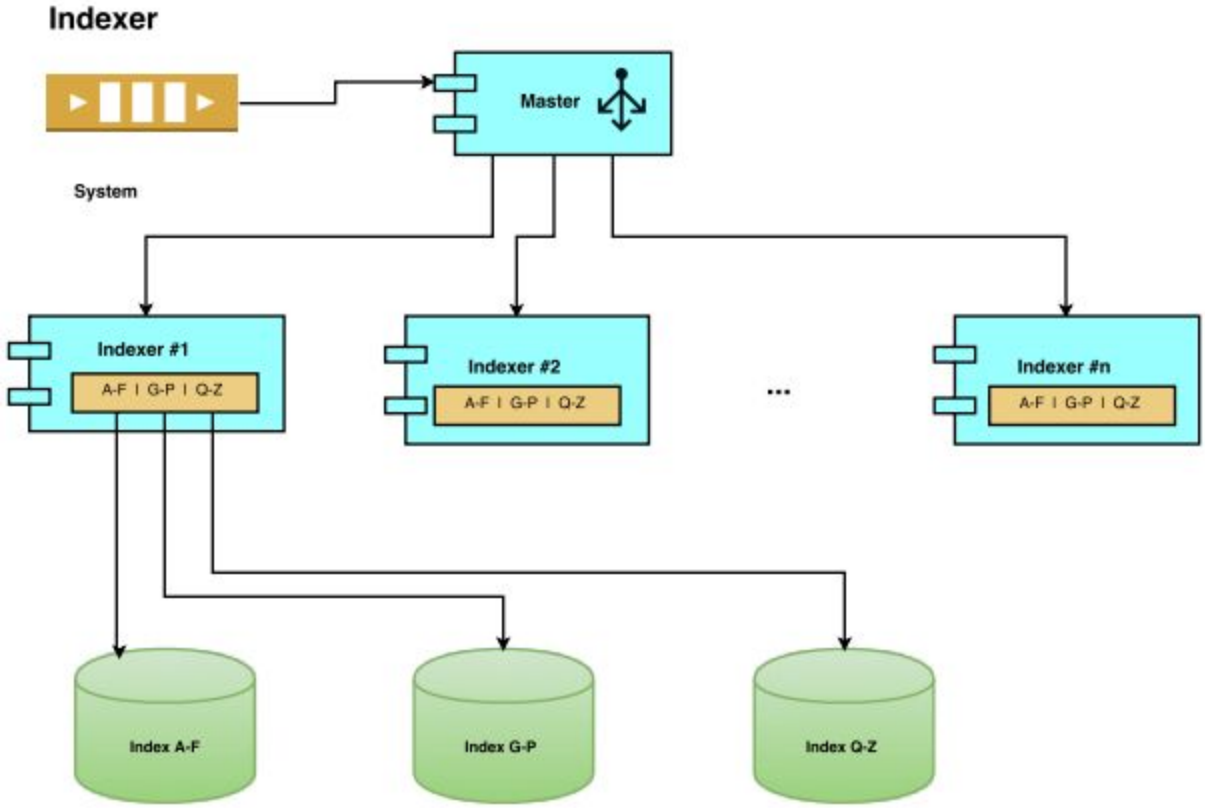
<[1,"Hello to the world. This world is BEAUTIFUL! "],
….>

output example for tokenizer:

["beautiful" -> [1,1].
"hello" -> [1,1],
"world" -> [1,2]]
….]

Indexer

Below is a more detailed architecture of the Indexer component of a search engine:



figure(4) High level architecture of Indexer

Tasks executed by the Indexer machine:

Title: Creates inverted index of all the documents in the corpus
Performed tasks:

- Create the inverted index
- Calculates tf-idf's (tf-idf= log(1+log(tf) * log (N/df)))
    - **output**: create a weight for each term in each index considering term frequency and how rare is the term in the corpus
- Normalize tf-idf
- Take care of an auxiliary index (for new documents in the corpus and deleted documents)
- Store indexes in a storage

Output: [key -> value]

Example Input of the indexer(Assuming N=3):

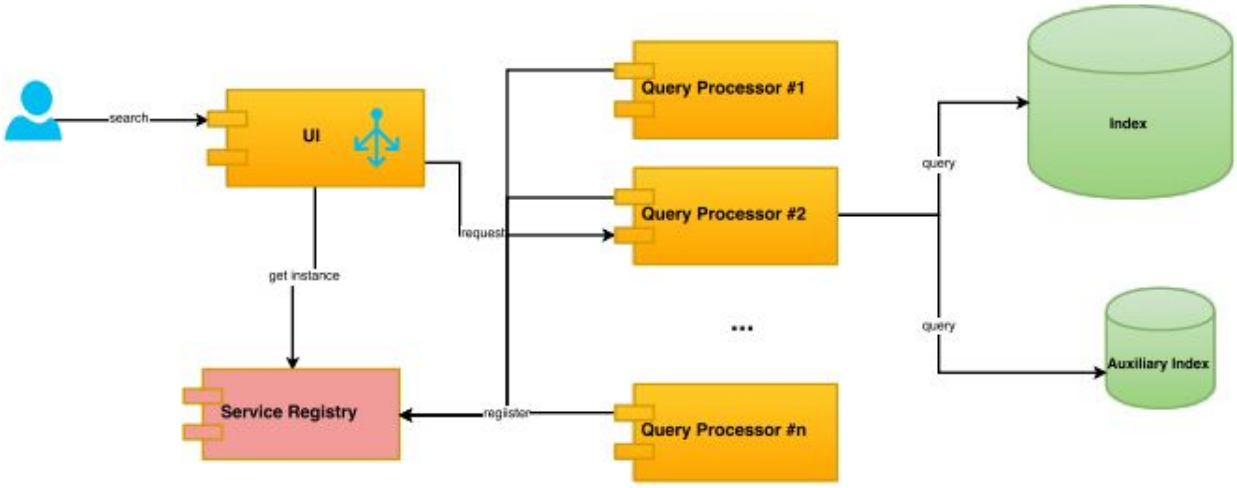
<
[“another” -> [1,1],“hello” -> [1,1],“world” -> [1,2]],
[“another” -> [2,1],“hello” -> [2,1]],
[“another” -> [3,1]] >

Example output of the indexer:

<[“another” -> [1,0],[2,0],[3,0].
“hello” -> [1,0.47],[2,0.47]
“world” -> [1,0.95]]>

Query Processor

Below is a more detailed architecture of the Indexer component of a search engine:

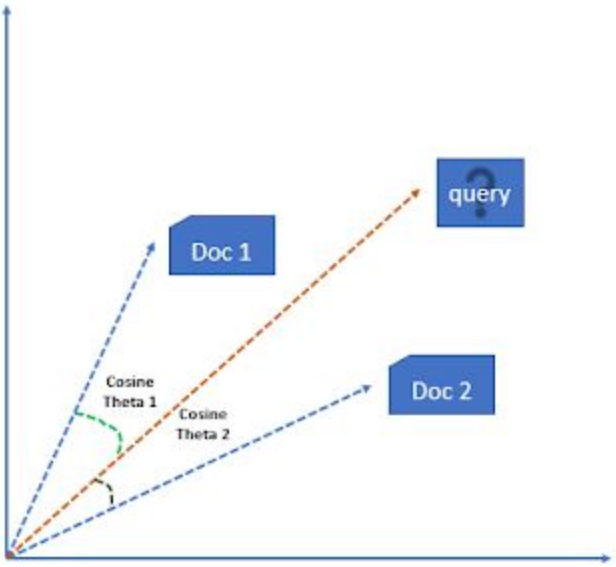


figure(5) High level architecture of Query Processor

Title: search for terms in the corpus
Performed tasks:

- Tokenize the query
- Calculates tf-idf’s of the terms in the query (tf-idf= log(1+log(tf) * log (N/df)))
- Normalize tf-idf
- Stemming and lemmatization (employee, employer, employment -> employ)
- Remove the stopwords (“a”, “and”, “but”, “how”, “or”, and “what)
- Apply soundex algorithm

- Provide relevance feedback
- Tokenization
- Search for the terms in the query (index and auxiliary)
- Intersect the results
- Find the cosine similarity of the query and relevant documents
    - cos(a)= (A.B)/(|A|*|B|)



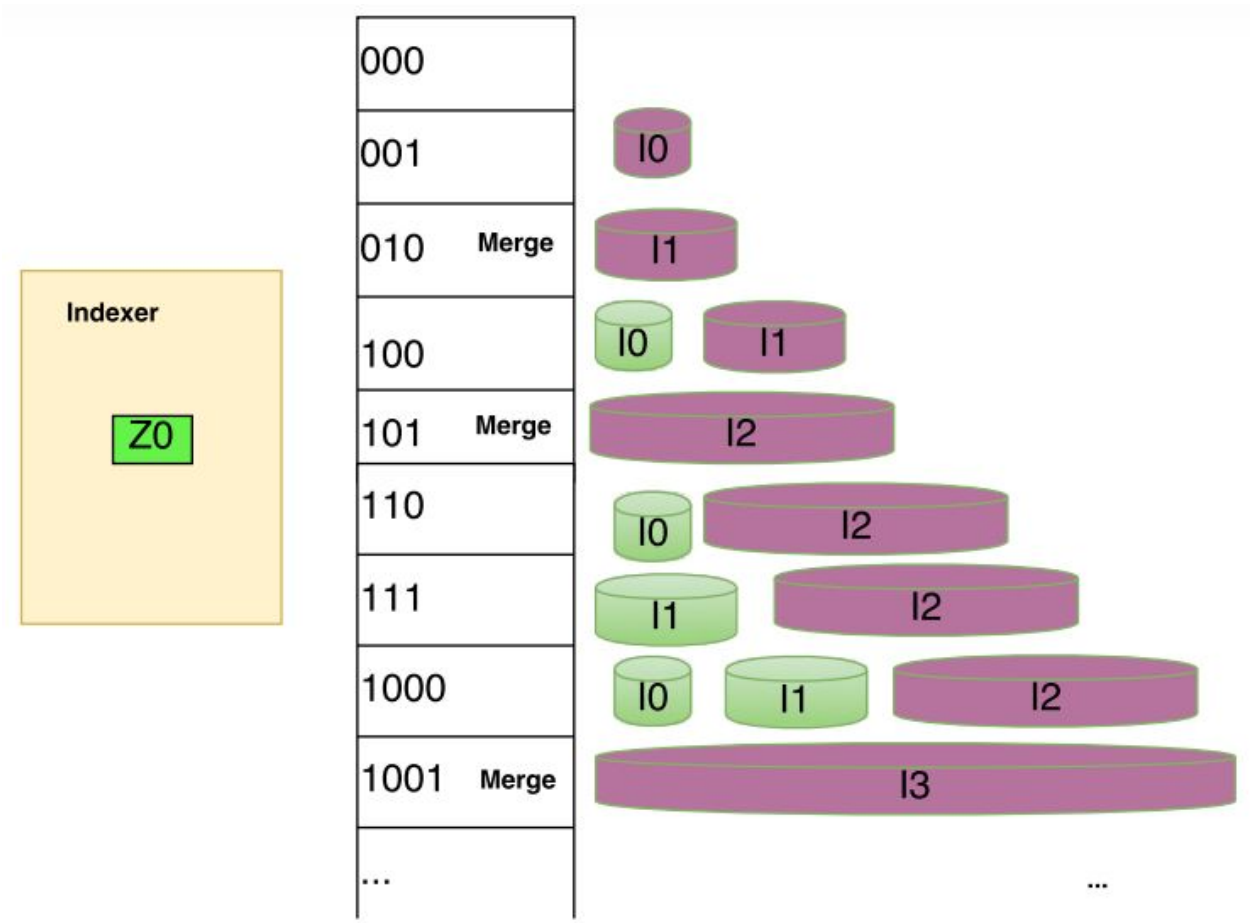
figure(6) Vectors of documents and the query

- Provide results

Output: relevant documents in the sorted order

Dynamic Indexing

In the target corpus, there are documents added and deleted every often. We keep track of those documents using dynamic indexing.
An auxiliary index will be created only for the newly added documents and query processor will get the results from both indexes when there is a query. The auxiliary index will merge with the main index in a logarithmic fashion. As indicated in figure 7 below:



figure(7) Dynamic indexing

Works Cited

"Information Retrieval Document Search Using Vector Space Model in R." *Data Science Central*, www.datasciencecentral.com/profiles/blogs/information-retrieval-document-search-using-vector-space-model-in.

"What Are Stop Words?" *Computer Hope*, 26 Apr. 2017, www.computerhope.com/jargon/s/stopword.htm.