Search Engine

Joel del Rosario, Victoria Torres, Álvaro Travieso, Alba Martín, Jorge Lang-Lenton Ferreiro

January 7, 2023

Abstract

Today, Search Engines are one of the most powerful tools of our generation, allowing users to locate information on the World Wide Web. Popular examples of search engines are Google, Yahoo! and MSN Search. But what makes them different?
Maybe it's the searching algorithm in your datamarts, or the specific treatment of these. Starting from an idea as simple as an online library, this project has wanted to reveal the true potential of these engineering that maintain the foundations of what we know today as Big Data. In this way, a personalized search has been achieved on a database of books, with their respective metadata, in which the user can filter, search and even obtain statistics on said content.

For all these reasons, we wanted to conclude with a global representation of the procedure for creating and scaling a simple Search engine project, demonstrating their potential and the complexity of their maintenance.

1 The answer to our questions: Search Engines

Each Search Engine uses different complex mathematical formulas to generate search results. The results of a specific query are then displayed on the SERP.
Search engine algorithms take the key elements of a web page, including the page title, content, and keyword density, and generate a ranking to place the results on the pages. Each search engine's algorithm is unique, so a top ranking on Yahoo! does not guarantee a prominent ranking in Google, and vice versa.
To further complicate matters, the algorithms used by Search Engines are not only closely guarded secrets, they are also constantly being modified and revised. This means that the criteria to best optimize a site must be met through observation, as well as trial and error, and not just once, but continuously.

In this way, those based on inverted indexes and SQL databases have been used as the main search algorithms in this model, since as it is a first approach, we have opted for the simplest but most effective cases in this type of searching technologies.

2 Problem statement

The problem addressed assumes great complexity. To do this, we define a class of concepts so that the reader can go to them at some point during the reading. Among these terms we will highlight inverted indexes, the use of SQL databases in this sector, which are containers or dockers, and the use of an API.

We will begin by explaining what an Inverted Index is in a simple way.
An Inverted Index is a system where a database of text elements is compiled with pointers to the documents which contain those elements.
Then, Search Engines use a process called tokenization to reduce words to their core meaning, thus reducing the amount of resources needed to store and retrieve data. This is a much faster approach than listing all known documents against all relevant keywords and characters. This type of datalakes has been used in a large part of the project's searches.

On the other hand we have the use of SQL databases.
Structural Query Language (SQL) is used for accessing, manipulating, and communicating with the database. Almost every function such as retrieving data from the database, creating a new database, manipulating data and databases such as insertion, deletion and updating can be performed using SQL.
Due to this, its incredible potential in this type of structure is remarkable. Among its great advantages we find Faster Query Processing, where no coding skills are needed. It is a standardized, portable and interactive language and it can provide multiple data views.

Continuing, the API maintains one of the most notable attractions in this project, its efficiency. When you have content that is automatically published and made available on different channels simultaneously, APIs allow for more efficient data distribution.

Finally, we will deal with dockers, which can simply be defined as a software platform that allows you to build, test, and deploy applications quickly.
Docker packages software into standardized units called containers that have everything the software needs to run including libraries, system tools, code, and runtime.
Due to the use of these, it has been decided to opt for the implementation of the so-called "load balancer", which acts as the "traffic cop" sitting in front of your servers and routing client requests across all servers capable of fulfilling those requests in a manner that maximizes speed and capacity utilization and ensures that no one server is overworked, which could degrade performance.

3 Method

3.1 Labor structure

The structure and design of this product has been developed as a group. Each member has been in charge of a part of the work, having a general approach to the final result. Among these parts, the work has been divided as follows:

- Victoria Torres Rodríguez: Persistence, testing SQLIte datamart creation.
- Joel del Rosario Pérez: Crawler, Core and benchmarking.
- Álvaro Travieso García: API Crawler and Inverted Index Structure. ´
- Alba Martín Lorenzo: Crawler and Inverted index structure file configuration.
- Jorge Lang-Lenton Ferreiro: Crawler (Controller) and Inverted Index.

3.2 Environment

Prior to the development of the experiments, it is necessary to explain where they were coded and executed for those who want to repeat and corroborate the results.

The program was implemented on the version 11 of Java, a highly common programming language for big data applications. Also, the tests were run on an M1pro chip, with a 10-core CPU, a 16-core GPU and 16 GB of RAM, so results may vary on other machines.

3.3 Methodology

- **SQLITE**

  Continuing with the methodology, the main datalakes used and their maintenance and creation are described below.

  First of all we must highlight the changes we have done on the Inverted Index, mainly related to the storage structure.
  Before this task, we used to have a folder for each letter and, inside them, we had a file for each word found on texts where you can look for the books where the word appears. This structure has as its main problem the need to access and read every word file if we want to get the books. This can lead to reduction in response times and unnecessary memory usage. For this reason, we change our file system structure.

  In this case, we have a folder for each letter, as before, and inside them, a folder for each word. This will compose a file for each book where the word appears. By this way, we never have to read any type of file and waste time required for it, we just have to get the title of the file.

  In the same way, for the effective resolution of queries that require a statistical base, an SQLite database has been chosen. This database will include a set of columns characterized by having a primary key that will be the "gutenberg id".

  In addition, it will include metadata of the corresponding books stored in the datamart, such as their title, publication date and authors, among others. For those values not found during the analysis, we have chosen to fill in with a NULL value, since in the future SQL searches that we will carry out in our API, this series of responses will not be useful when handling search errors.

  Next, the module in charge of this database will be briefly explained.

  First of all, we must establish a **controller**, in charge of managing the creation of subdirectories in which said database will be stored, and on which the data to be stored in the corresponding datamart will be searched.

  Like any other datalake, we must have a "manager" who will be in charge of establishing the corresponding connections with the database. In addition, it will allow the inclusion of new entries and close them. As the reader will have realized by now, there are some inconsistencies between Java and SQL data types. For this reason, it has been decided to previously generate a **processor** (with a functional interface), since the data to be entered into our database must be "clean", that is, that the search through it will be as easy as possible, if future mappings were involved.

  Once the data is processed, we only have to define the operations to be used. Being a simple project, we will implement the basic functions: add events, open connection, close connection, and read an event. For these, their respective interfaces have been created individually, with the aim of generating a database that is as consistent as possible.

- **API**

  The API module, as its name says, contains all the necessary classes for providing a properly built REST API, and also providing, in this case, certain responses to certain queries that have been selected with our own criterion.

  This module, firstly, contains a package called API (again), that divides itself into 2 packages:

  - **Operations**: It contains the classes that are related to the response/output that will be given to the API to a certain query. This package is also divided into 2 packages, each one representing a kind of response:
    * Stats: In this case, the response will give statistics about the queries given, that is, percentages and numeric terms about them, such as absolute frequencies (counter), relative frequencies (proportion), etc. Furthermore, to obtain these statistics, our program will need to access some datamart, being, in this case, the SQLite database that will have been created. For this task, the program will be using the QueryMetadataDatabase class, which implements QueryDatabase.
    * Words: In this case, this package is not focused on statistics (as it is Stats), but more on giving the words and books that meet certain conditions. As well as in the Stats package, these classes will need a datamart from which to obtain these characteristics and compare them. So, in this case the datamart will be a group of packages containing ".tsv" documents, representing the books that contain certain words and the lines where they are, what we know as the inverted index. These datamart will be accessed with the DocumentSearcher.

    Both of them will have a Controller class that will move the threads of what the classes need to do at each moment, as well as different Output formats depending on the response we want to give.

  - **Service**: It will be in charge of starting the web service (the main program starts here) with the API REST being the one who will give the responses to the queries. In this case, the API will obtain the information given by the datamart obtained through GutenbergBooks, so an API implementation for this datamart has been created. This package only has a subpackage:
    * Parameter: This package is related to the parameters that represent the query created by the user in the web service. This is, a class for obtaining the parameters in the query and splitting them into primary (that will determine the kind of response returned) and the other ones, as well as a class to store them (Parameters).

  - **Utils**: A package with classes to access the different datamarts, that the previously mentioned classes will need in order to give the responses. There are 2 packages included in this one:
    * Database: This package includes the necessary classes for filtering the books that meet certain conditions related to the metadata, this is, QueryMetadataDatabase, that implements QueryDatabase, as well as AnswerMetadataFilter, that implements AnswerFiller. These 2 last ones will help us to build the result iteratively. These classes will make it possible for the ones included in the "stats" package to obtain the information in the SQLite database.
    * Filter: It contains, as its name says, the necessary classes for filtering the books that meet certain conditions related to the words, this is, GutenbergDocumentFilter, that implements DocumentFilter. These are the classes that the "words"

package will need, accessing, this way, to the inverted index.

Finally, a class called JsonUtil has been created to make it possible to send the response to the web service in a good looking JSON format for the user to be pleasant and easy to understand.

- **Docker**

  As it was said before, another objective of the task was to dockerize the whole project.

  To achieve this goal, we had to follow some steps which will finish with creation and deployment of a docker container. First, we started with the creation of a Dockerfile for each module created on our project. All of them share the structure of the file and differ on the names used as we will later.
  A Dockerfile refers to a text file that contains instructions about how to build a Docker image and allow us to automate the process of creating it. In these files we may include instructions to indicate the basic configuration of the image created. In our case, we use the next ones:

  – FROM: This instruction specifies the base image that the Docker image will be built on top of. As it is show on the code, we use the image of OpenJDK.
  – COPY: This instruction is used to copy files from a source on the host file system into the image. We use this instruction to copy the jar file of each module to docker.
  – WORKDIR: Used to specify the working directory for subsequent instructions in the Dockerfile. In our case, we establish as the main directory the location of the jar files.
  – CMD: This instruction is used to specify the default command that will be run when a container is launched from the image. In our case, it is used to launch the jar file.
  – EXPOSE: This instruction is used to specify which ports should be exposed by the container when it is launched. This was a very important instruction on the API Dockerfile to be able to use the 4567 port with docker.

  These Dockerfiles will be responsible of creating the docker images, but we still need a mechanism to execute all of them. To solve this, we have docker-compose, which is a tool for defining and running multi-container Docker applications. It allows you to use a YAML file to configure the application's services, networks, and volumes. In our case, we will use docker-compose to run every Dockerfile and declare the volume used, which is the disk directory.

4 Conclusions

As we have been able to see, the development of the Search Engine means a really useful service for other people.

Creating a simple online library for people to search for books that meet certain conditions, as well as looking for statistics about these books contained in the library, we have also been able to see the wide range of possibilities that Big Data can reach, not only in this area, but also in most job sectors.

If we were to reach some conclusions about this project, we must outstand the importance of the proper distribution in modules, packages and classes that represent the project, that is, the project structure, for a better understanding of the project.
Also, another thing to outstand would be the importance of using a load balancer, in this case with Nginx. As we have seen, when trying to use the web service only provided, in this case, by a computer, it is harder for it to distribute the responses to certain queries than if we were using

the load balancer between more computers, making it easier to answer the queries.

5 Future works

Even if this work meets the requirements that it had, there are some facts and aspects that can be improved. So, for future versions of this Search Engine, these might be some changes that could be implemented:

- The creation of a better user interface, in terms of interactivity and beauty

  We have created an interface that simply responds to the queries that the API receives through the searching bar, but it would be, obviously, easier for a user if we gave them some deployable bars where they could select what to use, instead of making them write what they want.

  Not only that, but also that if we want people to use our service, we have to make it attractive for the user to see.

- Other improvements in the code

  The perfect programming code doesn't exist. There is always something that can be improved in terms of cleanliness, readability, memory or even performance.

  Even the project structure could probably be improved. The classes, packages, interfaces and modules could have probably been done better, as well as their names.

  But, in the end, it's a matter of what kind of people are working on the project, their thinking, experiences, etc.

6 References