ULPGC · UNIVERSIDAD DE LAS PALMAS DE GRAN CANARIA

BD

Sparse Matrix: big problems require compressed solutions

Álvaro Travieso García[1]
[1] *alvaro.travieso101@alu.ulpgc.es*

Alba Martín Lorenzo[2]
[2] *alba.martin112@alu.ulpgc.es*

Victoria Torres Rodríguez[3]
[3] *victoria.torres101@alu.ulpgc.es*

Joel Del Rosario Pérez[4]
[4] *joel.del101@alu.ulpgc.es*

Jorge LangLenton Ferreiro[5]
[5] *jorge.lang101@alu.ulpgc.es*

Abstract

The importance of web crawlers in the data age in which we live, has gained great importance in the development of software solutions for Big Data-oriented problems. Web crawlers, or search engine bots download and index content from all over the Internet. The goal of these bots is to find out what (almost) all web pages are about, so that the information can be retrieved when needed. They are known as "web crawlers" because crawling is the technical term for automatically accessing a website and obtaining data using a software program. Once the indexing structure is established in a Big Data project, it must be tested with a real problem. For this reason, this work has focused on the creation of a Web Crawler on which the foundation of an online library will begin to be laid, based on the "search-engine" methods. Al lof this Project has been executed following a TDD methodology and SOLID and polymorphic design patterns. Said experimentation has been executed on different types of arrangements and methods, taking into account the time elapsed in its execution in seconds. For all these reasons, it has been concluded that execution times are relevant in this type of project. However, despite the fact that Python maintains relatively better times than Java (creation of an inverted index of 10 documents in 2,000 milliseconds compared to 15,000 that Java takes), the latter allows for a scalable product structure, being easily modifiable in the future. future.

Key words: *Crawler, Index, Persist, Metadata, Book, Gutenberg, Deserialize, JSON, TSV, Inverted Index.*

1 Web crawling, today's golden tool

A web crawler, web crawler or spider, is a computer program used to automatically search and index website content and other information across the Internet. These programs, or bots, are most commonly used to create entries for a search engine index. On the one hand, there is the crawler, which is the core for downloading the files, archives or web content to be indexed. It is normally carried by an API. On the other hand, there is the search indexing. If we want to create a visual comparison of what indexing is and what it is for, we will simply have to imagine it as the creation of a catalog of library cards for a system so that a search engine knows where to retrieve information on the Internet or another system when a person makes a search. search. At present it has become an important part of human life to use the Internet to obtain WWW (World Wide Web) input data. The current population of the world is approximately 7.049 billion of which 2.40 billion people (34.3 percent) use the Internet. For this reason, numerous studies are currently being carried out to achieve the fastest search engine (Summary of web crawler technology research, Linxuan Yu et al 2020). For all these reasons, our study wanted to highlight the importance of good development of this type of product, comparing the speed of languages (Python and Java), the chosen inverted index structure, and the storage of data in a datalake.

2 Problem statement

This problem may seem complex, so we will define in a simple way the main components of this study. A crawler is a computer program used to automatically search and index website content and other information across the Internet. This type of software is very widespread among programmers due to the facilities it provides for working with information and data from web pages. The crawler structure also includes the treatment of the downloaded information to adapt it to our needs. These types of software are defined by the speed of and obtaining and amount of data downloaded, being able to determine the success or failure of a program. For example, Amazon and Google have their own crawler and it's a key part of their company, following a structure described in Figure 1.

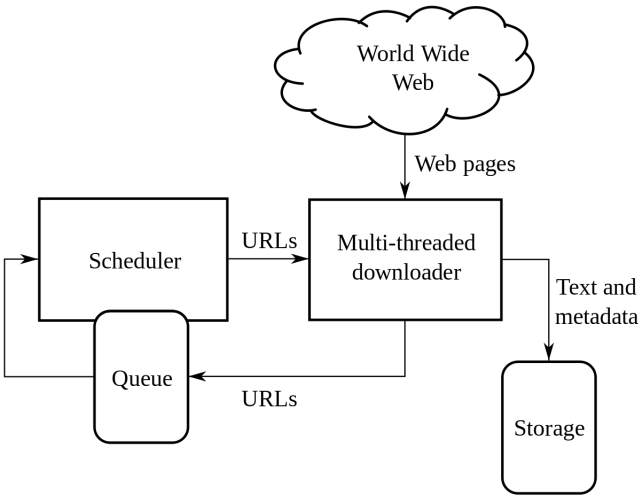

Figure 1: Web Crawling process

The second part of the project is focused on the development of an inverted index, which has been explain on previous studies. However, in this case we will carry out the implementation on Java, which allow us to compare the performance of this structure in both language. More specifically, in this paper we will explain the procedure followed to create a crawler of the Gutenberg books available on the web page and its subsequent storage on disk as parsed objects. These documents can be recalled from memory to create an inverted index them, which is the final aim of this project as we will see along the article.

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 and benchmarking.
- Joel del Rosario Perez: Crawler, Core and benchmarking.
- Álvaro García Travieso: API Crawler and Inverted Index Structure.
- Alba Martín Lorenzo: Crawler and Inverted index structure file configuration.
- Jorge LangLenton 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 others machine.

3.3 Design

The design of the code it´s an important part of the program. In this case, we divided the production code into three different modules. The first one is called core and includes all the classes used by the rest of modules, like Document and Metadata. The rest of the modules will implement this code by a reference on the pom file.

The second module is crawler and contains all the implementation related to the good performance of our crawler. As you can see on the code, we have 4 different packages on it. Controller includes the classes required to read the configurations file and start the crawler. The second packages is web_connection and is the responsible to download the information from the requested book. Also, we have document$_b$*uilderwhichreceiveastringandreturnadocumen*t

Inverted$_i$*ndexisthethirdmoduleimplementedandiscompoundby*3*packagesneededtoimplementthefunctionalityoftheinve*

3.4 Methodology

Prior to implementation, a class diagram has been created in StarUML to represent the fundamental objects of the system that will be part of the solution and the relationships between the elements of the system, in order to see what components the system needs and how they influence each other. For this purpose, the SOLID methodology and the polymorphism of the classes have been taken into account. First of all we have divided the project into three modules: crawler, core and inverted-index.

1. The crawler module is responsible for making a web connection and load the content of books, from which generates a folder whose name is the download date, within this a folder with an id associated with the randomly generated book and within this a file with json extension that corresponds to the metadata of each book, a file with txt extension that corresponds to the content of the book and another file with txt extension with other data of the operation. As we can see in Figure 2, the module is divided into five folders:

   - controler: to start the execution with a given timeout between each iteration.
   - document_builder: in charge of extracting the metadata and the content.
   - file_system: it is in charge of creating the folders and files.
   - id_treatment: it is in charge of creating files to store the ids of books that have already been downloaded or that have already been tried to download but do not exist.
   - web_connection: it is in charge of connecting with the web of the books and load the content of these.

2. The inverted index module creates an inverted index from the folders generated by the crawler module, so that a tsv extension file is created for each word that will be stored in a corresponding folder with its initial and whose content will be the first column the book in which the word appears and the second a list with the lines in which the word appears.

   - document: contains essential classes for deserialization, creation and persistence in data memory.
   - program: contains a main inverted index control class and another one to eliminate non-relevant characters from the books.
   - inverted_index: this folder is divided into two subfolders.
     - implementation: it is in charge of creating the inverted index.
     - storage: it is in charge of creating the folders and files on disk.

3. The core module contains the common crawler and inverted index classes. These classes are Document, which initializes the document attributes; Metadata, which initializes the metadata attributes; and FilePersistor, which receives a file and a text and writes the text to the file. Finally, we have a configuration file that contains data such as id_init (from which id to download), iterations (total number of books) and source (from which web) and delay (time that passes before starting to execute the first iteration).

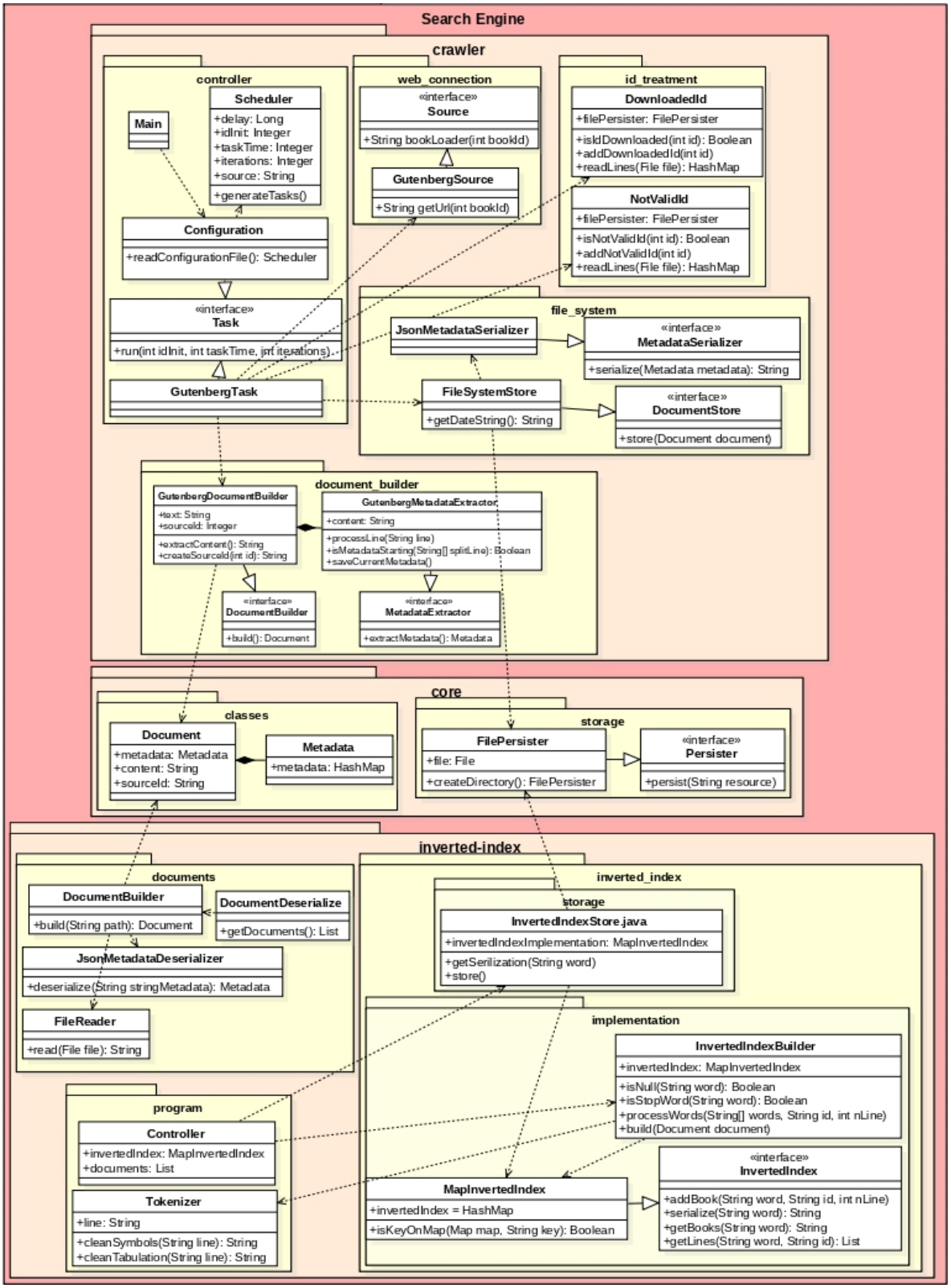

Figure 2: Class diagram for crawler, core and inverted index implementation

4 Experiments

The tests that were executed and the results that were obtained in said analysis will be demonstrated. To evaluate the crawler book loading and the inverted index implementation, it has to be must measured different aspects of performance in order to establish if these solutions are optimal.

These experiments have been developed under JMH, a Java harness for building, running, and analyzing nano/micro/milli/macro benchmarks written in Java and other languages targeting the JVM. All of them have been developed under the same circumstances and the same type of computer, described previously.

4.1 Experiment 1

One of the cores in text indexing systems are tokenizers. For this reason, we wanted to carry out a series of tests on different cases to verify the effectiveness of this class. Among them, the elimination of characters other than punctuation marks, exclusive tests for punctuation marks, elimination of numbers and general tests that mix all these characteristics were verified. All these tests were executed successfully. However, there is a series of words that could not pass a test that we will discuss later related to the inverted index. Windows systems maintain a series of files with special names (ON, PRN, AUX, NUL, COM1, COM2, COM3, COM4, COM5, COM6, COM7, COM8, COM9, LPT1, LPT2, LPT3, LPT4, LPT5, LPT6, LPT7, LPT8, and LPT9). For example, writing to AUX on WindowsOneDrive causes writing to Auxiliary device, usually a serial port. Due to this, in the tokenizer these types of words have had to be treated differently.

Another core of this system is the detection of "stop words" which have been treated through a comparison between the text given as input and a TXT that contains a set of said words. In the same way as in the previous tests, these were successfully passed.

4.2 Experiment 2

Another point to highlight in this project is the download times of different books. For this, a benchmarking has been carried out with the download times of a certain number of books, taken in milliseconds. As can be seen in Figure 3, it seems that the growth over time as the number of downloaded documents increases is linear, a desired feature in this type of software product, so we can determine that the crawler works properly.

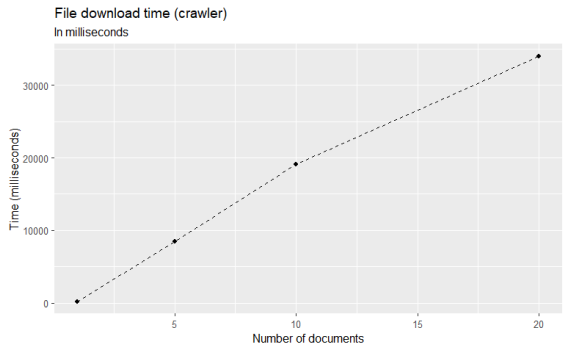

Figure 3: Crawling downloading time

4.3 Experiment 3

Moving on from file download times, we will now discuss the creation and persistence time of inverted indexes. As can be seen in Table 1, times per book are usually 3,000 milliseconds, maintaining an almost linear growth growth, as was desirable. On the other hand, the comparison of Python and Java has shown that despite gaining speed with Python creating an inverted index of 10 documents with an average of 2,000 milliseconds, compared to 15,000 for Java, it can be verified that a system has been added. of persistence. and a new structuring of the indexes, creating them individually. For all this, we have gained a great general consistency of the product, since future searches will be more effective and faster, compared to python.

Number of books	Mode	Iterations	Execution Time	Error	Units (ms/op)
1	avgt	3	2,972.363	2,842.507	ms/op
5	avgt	3	6,459.895	11,449.132	ms/op
10	avgt	3	14,953.332	6,789.056	ms/op
20	avgt	3	36,692.187	97,833.998	ms/op

Table 1: Benchmarking of creation of inverted index of n number of Documents.

5 Conclusion

As we have seen, search indexing is like creating a catalog of library cards for the Internet so that a search engine knows where to retrieve information on the Internet when a person does a search. This development system is intrinsically supported by crawling, algorithms and APIs for downloading web content. Together both concepts, we managed to develop a software product that will be the basis of future Search Engine projects.

As it has been exposed, our project consists of three differentiated modules, a core that contains those classes used in most of the project, which refer to data about the downloaded documents. On the other hand, we have the module referring to the inverted index, which will contain those classes that facilitate and help its creation and persistence: InvertedMIndexBuilder, DocumentBuilder, JsonMetadataDeserializer, Tokenizer, etc. All this will result in a datalake referring to the words of each analyzed document, with subfolders whose name will be the words of the alphabet, in which we will store the index individually of each word in TSV. Finally, as the last module, we have the crawler, in charge of downloading the documents, serializing their metadata in JSON format, and their content in TXT format, generating in turn a new datalake where these files will be stored, whose folders will be an id generated from random way. In turn, said id will be saved in configuration files, to avoid re-downloading these books again.

As we mentioned in the experiment, the speed of the download crawler seems to be linear, taking an average of 200 milliseconds for each book. On the other hand, the comparison of Python and Java has shown that despite gaining speed with Python creating an inverted index of 10 documents with an average of 2,000 milliseconds, compared to 15,000 for Java, it can be verified that a persistence system has been added and a new structuring of the indexes, creating them individually. For all this, we have gained a great general consistency of the product, since future searches will be more effective and faster, compared to python. The main contribution of this study is based on the approach to search engines from a "junior" point of view sight, coming to understand firsthand how this kind of engineering works and where its importance lies: in the ability to create future implementations, maintaining good persistence systems and index structuring. In addition, it has been verified how the change of programming language achieves that there are more desirable characteristics, such as greater persistence and general structuring of the program, which will facilitate future implementations of any type. As a first contact, it has been understood that indexing is one of the main and most common methods in this type of problem, allowing to appreciate and understand from an initial point of view, how to develop a search software product, starting with its core. We have also verified that these methods go hand in hand with the so-called "crawlers", which will be the main source of data to be indexed or processed in order to carry out the indexing methods.

6 Future Work

Many different adaptations, tests and experiments. they have been left to the future due to lack of time (ie experiments with real data are often very time consuming and require equal days to complete a single run). For all these reasons, we would like to name some aspects that would be taken into account for a better implementation of this study. First of all, we must recognize that there are points for improvement that should be addressed in future papers. Fox example, different tests were left out due to the lack of time and resources. Also, we have to take into account some improves that can be implemented on the inverted index, like the recognition of plural words that would allow us to extract better the most important words of the book. Also, we must highlight that different languages inverted index can be implemented as well, aside from English to achieve a more complete project. Finally, we must mention that more types of serialization should had been added to the code, like JSON for the inverted index. In this way, we would be able to compare which one is better and make the right choice.

References

[1] *Summary of web crawler technology research* Linxuan Yu. 2019.

[2] *How to create CON, AUX, NUL folder and files with a click in Windows OS* Anand Khanse 2018