Search Engine Project

Inverted Index

Adam Breznen[1], Ricardo Juan Cárdenes[1], Joaquín Ibáñez[1], Mara Pareja[1], and Susana Suárez[1]

[1]Faculty of Computer Science
[1]Las Palmas de Gran Canaria University

October 7, 2023

Abstract

In this article, we present an approach to efficiently create an inverted index from text documents using the Python programming language. We aim to address the challenge of building an inverted index from large textual datasets. Initially, we considered two possible approaches with the goal of achieving maximum speed and efficiency in execution. Consequently, we discarded our initial idea, which was to use a relational SQL database fed from a full-book datalake, after a brief investigation. Instead, we opted to separate the content and metadata of such free books to implement a NoSQL database using MongoDB, which led to a noticeable increase in execution efficiency and speed. The results derived from our experiments support the effectiveness of our solution. Additionally, we analyzed future enhancements involving the use of metadata to enrich the search capabilities and considered the possibility of implementing the system in Java in order to increase speed as well as automate the downloading of documents from our data source, "Project Gutenberg".

Keywords inverted index, non-relational database, search engine, project guthenberg

1 Introduction

Generating an inverted index from text documents is an essential component in structuring and efficiently searching for information in today's vast digital landscape. In this article, we meticulously investigate an approach to address this undertaking, making use of Python and non-relational database technologies, in particular, MongoDB.

The need for an efficient inverted index in information processing and crawling is not new. It has been a mainstay in information retrieval, search engines and word-processing applications. However, in an environment characterized by constant evolution and the proliferation of digital data, efficiency and speed in the design and use of these indexes are of paramount importance.

The problem we address in this research focuses on the urgency of creating an efficient system capable of processing vast volumes of text and enabling agile and precise searches in these documents. To this end, we asked ourselves whether a NoSQL database, such as MongoDB, could represent a more effective alternative in terms of performance compared to traditional relational databases.

The main contribution of this study lies in the successful implementation of an approach that amalgamates metadata and content segregation, text normalization and adoption of such a database, highlighting its efficiency and speed in the conception of an inverted index. In addition, this research lays the groundwork for future developments, such as the refinement of metadata search and eventual implementation in Java to increase speed and automate document downloading. These collective contributions have the potential to have a substantial impact on the efficiency of searching and organizing information in contexts characterized by the abundance and diversity of textual data.

2 Problem Statement

The challenge lies in effectively organizing and searching the information contained in a large number of text documents. As the volume of digital data continues to grow, there is a pressing need for a method to quickly and accurately index and retrieve information from these documents. The creation of an efficient inverted index presents itself as a fundamental solution to

address this problem. The problem to be solved is broken down into several key aspects:

- **Document segmentation and construction of a Datalake.** It is essential to develop an approach that enables the division of documents into metadata (descriptive information) and textual content of the book. This separation is indispensable to achieve an organized and efficient structure in the Datalake, enabling effective search and retrieval of information in an ever-expanding digital environment.
- **Text normalisation.** It must be ensured that the text is normalized so that searches are insensitive to differences in capitalization and irrelevant words are excluded.
- **Efficient storage and access.** The organization and database used must be efficient to allow fast searches and accurate information retrieval.
- **Scalability** The solution must be scalable to handle large text datasets.

3 Solution/Methodology

The solution proposed in this study addresses the task of generating an inverted index from text documents in an efficient and scalable way. Our approach combines several essential steps that enable the creation of this index effectively and facilitate its replication by other researchers.

3.1 Metadata and Content Division

The process starts with the segmentation of the documents into two key components: metadata and textual content of the books. To achieve this, a function was developed that identifies a closing and opening sentence. The metadata is stored separately in a directory called "Metadata", while the textual content is stored in a directory called "Content". This step allows a clear distinction between the descriptive information of the document and its main content from which we will create the inverted index.

3.2 Text Normalization

To ensure uniformity and consistency in the inverted index, we implemented text normalization. This process involves the conversion of all letters to lowercase, the removal of stopwords (common words that do not contribute meaning), and the exclusion of single-letter words as well as irrelevant special characters. The NLTK library, Python's Natural Language Toolkit, was used to carry out this normalization. This step is crucial as it ensures that searches are insensitive to capitalization differences and eliminates irrelevant terms.

3.3 MongoDB database

In terms of database management, the project explores both relational and NoSQL databases. The initial implementation employs a relational database schema, but the team also evaluates MongoDB, a NoSQL database. This database stores key information, including an inverted index that relates words to the documents in which they appear. We finally chose MongoDB, this decision was based on performance and efficiency tests, which indicated that this technology provides a faster and more scalable solution compared to conventional relational databases.

4 Experiments

On the one hand, we would like to highlight the decision to split documents into "Content" and "Metadata" directories, rather than continuously analyzing the document structure to identify metadata boundaries. This approach, involving upfront separation of metadata and data a scalable option going forward rather than the second option, processing the documents "raw" each time to locate the metadata sections.

On the other hand, the experiments conducted in this project to measure the execution time of different types of database technologies (relational, and non-relational) cover two main areas: creation and insertion time for different document counts.

4.1 Database creation time

In the first set of experiments, we evaluate the time required to create an empty database for both SQL and NoSQL database management systems. Specifically, we aim to measure the time it takes to initialize a database without documents. This benchmarking process is crucial to understand the benchmark performance of each database system when there is no existing data to process. We repeated this experiment for both SQL and NoSQL databases, allowing a direct comparison of their creation times. In this scenario, the NoSQL database showed superior performance, consistently outperforming the SQL database in terms of database creation time. This observation underscores the advantage of NoSQL databases

for quickly creating data warehousing systems when starting from scratch.

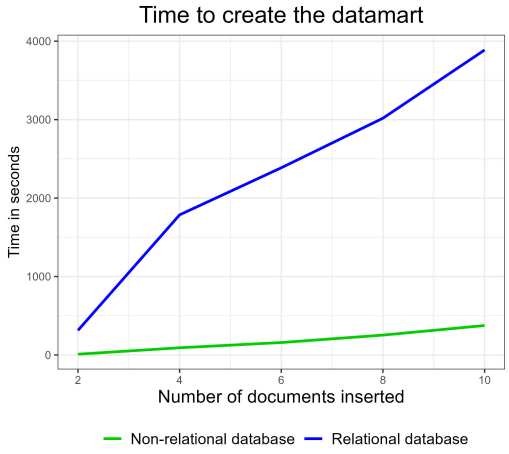

Figure 1: Replication Based Technique

We conducted these experiments with different document counts, from two to ten text files, which allowed us to observe how the creation time of each database system increases with document count.

4.2 Insertion performance

Now, we focus on the performance of inserting documents into an already initialized database for both technologies. To do so, we start with an existing document and measure the time it takes to insert it into the 10-book database created for the previous section 4.1. We continue this process incrementally, inserting two, three, four, and five more documents, in pursuit of measuring how much time it takes to update our data-mart depending on the number of files that are going to be indexed.

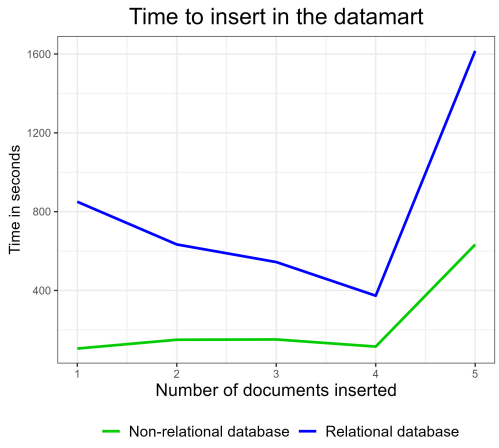

Figure 2: Datamart Creation Time Comparison

Consistent with our results from the database creation time experiments, the non-relational database demonstrated higher insertion performance. This trend was evident in all phases of document insertion, highlighting the efficiency and scalability of NoSQL systems in accommodating growth and data updates in a dynamic environment. These results reaffirm the advantages of non-relational databases in scenarios requiring fast and efficient data manipulation and updates.

The charts introduced in this section, despite showing how can MongoDB outperform document indexing, do not show the real behavior of this technology. If we take a look at Figure 2, it may seem that inserting five documents is much worse than inserting just four. Moreover, it looks like inserting four files is less complex than inserting three. The explanation that underlies this phenomenon is that the implied files do not have the same size in terms of memory for each step. Thus, inserting one single document could be even slower than inserting five, if the first one is bigger than all the others together.

To show how fast can MongoDB index documents, we give the reader Figure 3, built up using the same methodology as for Figure 2, where it can be seen how long the insertion takes for several numbers of documents of the same size.

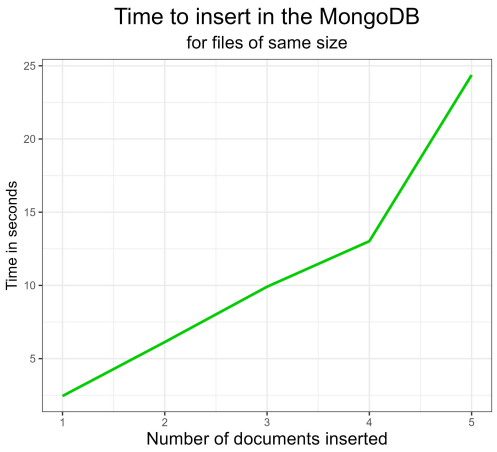

Figure 3: Datamart Insertion Time Comparison

Just as expected, we obtain a linear growth of the insertion time as the number of inserted documents gets bigger, at least for the first four amounts of them. What happens when inserting five documents is that two of those are written in another language, so they take longer to be inserted, as they have to insert more information into the database. Now, these results do not prove that the insertion does not take longer when the database grows, as we are not removing the inserted files for each step. Even though

one could assume that state due to the linear behavior of the insertion time, it should be pretty clear that the slope of the function in terms of the number of files is exactly one. To prove this claim with no suppositions, we must show how for those same files the number of inserted words per second remains the same.

Figure 4: MongoDB words indexed per second

As the reader can appreciate, linear behavior in insertion time does not imply constant conduct of inserted words per second. Anyway, since we do not know the word distribution of each document, we can not tell that this decrease in performance is due to the size of the database, disproving our stated claim, as it could be caused by the differences between the index content and the files to insert.

5 Conclusion

This project presents a comprehensive approach to efficiently create an inverted index for textual data. It highlights the importance of metadata extraction, text normalization, and database selection to improve retrieval performance.

The separation of metadata from data and text normalization contribute to accurate and efficient query processing. In addition, benchmark results favor MongoDB as the database management system of choice, due to its superior run-time performance, ensuring fast information retrieval.

This project not only addresses the specific challenges of creating inverted indexes but also provides valuable insights into best practices for effectively managing and querying textual data.

6 Future Work

Future work on this project will explore the use of metadata to improve search capabilities. By leveraging stored metadata, the system can provide more context-sensitive search results and improve the user experience.

In addition, the project aims to implement the process of creating inverted indexes in Java to take advantage of its performance benefits. Moreover, a new phase of the project will focus on automating the retrieval of text documents from the "Project Gutenberg" website, expanding the scope and utility of the inverted index repository. By this task, we will actually be able to make conclusions on the performance of MongoDB for indexing large amounts of documents, as we could not in section 4.2.

On the other hand, a promising direction is the development of an API (application programming interface) that would enable seamless integration of the inverted index repository into various applications and services. This API could facilitate programmatic access to indexed data, allowing developers to create customized search and retrieval functionalities tailored to specific use cases.

Finally, another idea we had is that incorporating machine learning algorithms for text classification and sentiment analysis could enable the system to provide deeper insight into document content. This feature could be especially useful for applications such as content recommendation and trend analysis.