International Journal of Internet, Broadcasting and Communication Vol.4 No.1 13-17 (2012)
http://dx.doi.org/10.7236/IJIBC.2012.4.1.13

IJIBC 12-1-4

PDFindexer: Distributed PDF Indexing system using MapReduce

JAziz Murtazaev[1], Jang-Su Kihm[1], Sangyoon Oh[1*]

[1,1*]*Department of Computer Engineering. Ajou University, Korea*
{muraziz, jangsu84, syoh}@ajou.ac.kr

Abstract

Indexing allows converting raw document collection into easily searchable representation. Web searching by Google or Yahoo provides subsecond response time which is made possible by efficient indexing of web-pages over the entire Web. Indexing process gets challenging when the scale gets bigger. Parallel techniques, such as MapReduce framework can assist in efficient large-scale indexing process. In this paper we propose PDFindexer, system for indexing scientific papers in PDF using MapReduce programming model. Unlike Web search engines, our target domain is scientific papers, which has pre-defined structure, such as title, abstract, sections, references. Our proposed system enables parsing scientific papers in PDF recreating their structure and performing efficient distributed indexing with MapReduce framework in a cluster of nodes. We provide the overview of the system, their components and interactions among them. We discuss some issues related with the design of the system and usage of MapReduce in parsing and indexing of large document collection.

Keywords: *information retrieval, parallel processing, indexing, MapReduce*

1. Introduction

Web search engines, such as Google, Yahoo, Bing allows searching a mass of unstructured hypertext documents across the entire Web. Those search engines index every semantically meaningful terms to enable keyword-based search. Besides that, there are many specific-purpose search engines providing searching capabilities on specific domain. Our focused domain in this paper is information retrieval on scientific articles in portable document format (PDF).

There already exist a number of scientific paper search engines, such as Google Scholar [1], Elsevier's ScienceDirect [2], IEEE Explore [3], and many others. They provide keyword-based search capabilities on structured scientific articles, which have metadata information such as title, authors, abstract, year of publication and etc. In contrast to those systems, we consider cases in which we have access to full-text articles in PDF without any additional metadata information. In such cases we need to extract that metadata information relying on semantics of scientific papers and the article layout specifics.

Most academic journals have electronic version of their publications in PDF. Making every article in every issue in every volume searchable is a challenging task because of large scale. Indexing large-scale data requires lots of computation. In order to perform indexing in effective way we need to use parallel processing techniques. Particularly, MapReduce framework has become popular in large data processing since it was introduced in 2003 by Google [4]. MapReduce framework enables doing parallel processing without worrying about distributed system issues, such as fault tolerance, scalability, reliability, replication.

We propose an Information Retrieval system, called *PDFindexer*, for indexing and querying scientific

Manuscript Received: Nov. 20, 2011 / Revised: Jan. 18, 2012 / Accepted: Feb. 13, 2012
Corresponding Author: syoh@ajou.ac.kr
Tel:+82-31-219-2633, Fax: +82-31-219-2633
Dept. of Computer Engineering. Ajou University, Korea

articles in PDF. Given large corpora of scientific articles in PDF our system parses and extracts article contents with additional metadata, such as title and abstract, and then indexes extracted content using MapReduce framework in a distributed system. The resulted indices enable making keyword-based queries on the corpora of scientific articles. Differences between web search engines and PDFindexer are given in Table 1.

In this paper we investigate the usage of MapReduce programming model for parsing and indexing scientific papers. We show architectural design and implementation plan of our system and left actual implementation and evaluation for future works.

Table 1. Comparison between Web search engines and PDFindexer

	Web search engines	PDFindexer
Data type	HTML pages (mostly)	PDF articles
Domain	web-sites of any content	scientific articles
Carry semantics	No, any text is treated equally	Yes, it has title, abstract, sections, references, authors
Major focus	Efficient indexing, crawling, ranking	Efficient indexing
Preprocessing needed	Typically no, only HTML tags removed	Yes, PDF should be parsed to text

The remainder of this paper is structured as follows: Section II describes background and related works. We show proposal of our system in Section III. Design details and implementation plan of the system is discussed in Section IV. Finally we conclude our work in Section V.

2. Background and Related Works

Our system relies on researches on such disciplines, as Information Retrieval, large-scale data analysis in distributed system, particularly, MapReduce framework.

A. Information Retrieval

Information Retrieval (IR) deals with finding documents of unstructured content that satisfies an information need from large collection of documents [5]. Information need is converted to query when it is submitted to computer which has the IR system. Documents are usually *indexed* in advance, which enables quick search, doing flexible operations such as applying Boolean operators AND, OR and other operations. And also indexing allows ranked retrieval of the documents based on relevance of the searched term to the returned document.

There have been several works related with information extraction from scientific papers. Briscoe *et al.* [6] propose a search engine for scientific literature. One of the features of their system is that they integrated text and image search which enabled fine-grained search for scientific papers. While they focus on improving search experiences for users, we focus on providing scalable indexing system. And they use Lucene [7] for indexing and retrieving results while we designed our own.

B. MapReduce framework

MapReduce is a programming model which enables specifying two user functions: *map()* which processes key/value pair and generates another intermediate key/value pair, and *reduce()* which merges all intermediate values related with the same intermediate key [4] (Fig. 1). This model came into being after realizing that developing parallel applications became very tough because of parallel computing issues, such as providing

fault tolerance, replication, availability, scalability. MapReduce model provides abstraction on all of these issues so that programmers can specify only two functions, and underlying framework takes care of parallel issues.

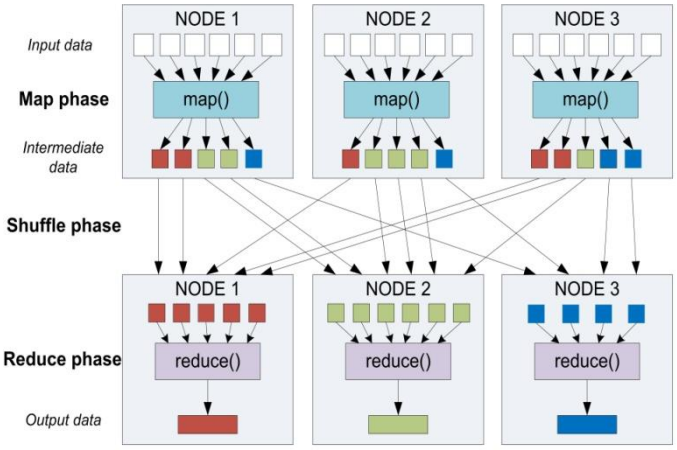

Fig. 1. MapReduce operation

After Google introduced this model, it became very popular both in industry and academia. Yahoo and Facebook have been sponsoring open-source implementation of Google's MapReduce model, named *Hadoop* [8]. Within Google MapReduce is used for such tasks as large-scale machine learning problems, clustering problems, large graph computations, web-pages properties extraction. MapReduce can be used for such tasks as inverted index construction, matrix multiplication, string matching, KMeans, linear regression and many others [9].

3. PDFindexer System Proposal

The overall ultimate architecture of our system is depicted on Fig. 2. PDFindexer accomplishes two tasks: indexing documents and querying on resulted indices. Those two tasks correspond to the two major components: *Indexer* and *QueryParser*. The workflow of the system is numbered separately for indexing and querying processes in this figure.

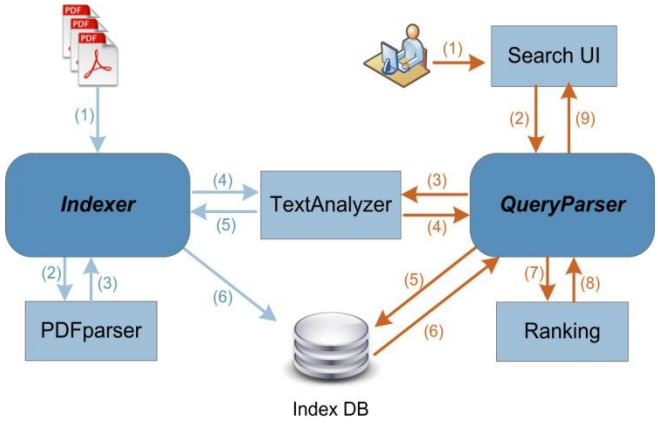

Fig. 2. Overall architecture of PDFindexer

In the indexing process, first, corpora of PDF files are provided to Indexer (1), and Indexer uses *PDFparser* subcomponent to parse PDF into plain text and extract context information as Document fields, such as Title,

Abstract, Body (2,3). Then plain text is sent to *TextAnalyzer* subcomponent to chop the text into small meaningful units of text – *tokens* (4). Besides just breaking text into tokens, TextAnalyzer performs additional job, such as removing very frequently used words, which carry little semantics taken separately, also known as *stop-words* (articles, prepositions, pronouns, numericals, etc), and also it extracts basic morphological form of the words for the purpose of grouping words with the same morphological origin ('jumping', 'jumped' -> 'jump'). As a result of these normalization operations we obtain a list of *terms* which is ready to be saved to DB. Indexer receives this list of terms along with their document mappings and frequency of the term in each field of that document (*posting-list*) (5) and saves them to the *Index Database* (6) where each entry will be represented as <term, posting-list> pairs.

In the querying process, user searching for some text (1), inputs it into *Search UI*, and it sends the query to QueryParser component (2). That query will be analyzed first by submitting it to TextAnalyzer (3,4), which breaks search text into tokens and then normalized, in the same way it was done in the indexing process. Next, those set of terms received from TextAnalyzer will be looked up in the Index DB containing <term, posting-list> pairs (5,6). For each term, posting-lists containing document IDs and frequency of occurrences of this term in this document are retrieved and scores calculated for each document appearing in the set of posting-lists (7,8). This is done in the *Ranking* subcomponent, and the results are displayed in the Search UI (9) as a set of document titles with the links to those documents.

This was a high level overview of the PDFindexer system. In single node mode, the interaction among the components is exactly as described in the Fig. 2. However, this architecture is not enough to understand how the system works in detail in large-scale clustered environments. We are going to parallelize the process, so that we can handle big scale in efficient manner. As input to our system is a large number of scientific articles in PDF, we need to consider how to split them into smaller pieces and process them in parallel. MapReduce framework takes care of all these steps, and additionally it provides job management functionalities, such as scheduling, monitoring and re-running failed jobs [8].

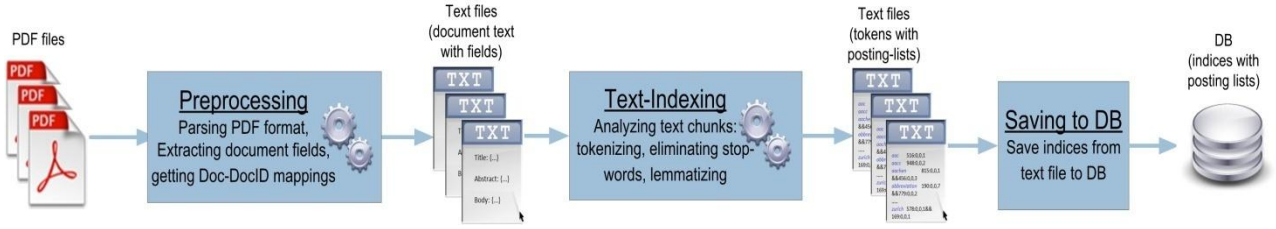

Fig. 3. Indexing process workflow

4. Design Details and Implementation Plan

Our ultimate design is affected by MapReduce programming model and distributed indexing concept described in [5]. Our implementation plan covers Indexer part only, QueryParser and its integration to our system is briefly discussed and will be included in future works. We will use open-source Hadoop MapReduce framework.

Our implementation of PDFindexer is currently undergoing and in this Section we show our implementation plan. Fig. 3 shows the indexing process workflow. We split indexing process into two parts: Preprocessing and Text-indexing. Although there is one more step in this figure, it only saves the result into DB.

A. Preprocessing large-scale PDF articles with MapReduce

Preprocessing step first creates global document to documentID mappings. In MapReduce framework, map tasks do not share context, therefore we need to obtain global mapping before MapReduce job. Preprocessing MapReduce job mainly parses large number of scientific articles in PDF in parallel and extracts document fields (title, abstract, body). In this MapReduce operation we have only map phase. In this case, the outputs of map tasks are written directly to Hadoop Distributed Filesystem (HDFS). Map tasks

process <docID, pdf_contents> key-value pairs as input. Output of map task is <<docID, documentField>, contents> pair, where documentField is enumeration value (one of three values: Title, Abstract, Body) and contents is the text content of the corresponding field.

B. Text-indexing of parsed articles with MapReduce

Text-indexing process is another MapReduce operation. Map() function takes <<docID, documentField>, contents> key-value pair (outputs of previous MapReduce job) as input and text contents are analyzed to produce a set of terms as described in Section III. Output of map() function are <term, <docID, documentfield>> key value pairs. Reduce() function creates posting-lists of each term by counting term occurences in each field in each document. Output of reduce task is <term, posting-list> key-value pair in this format: *"throughput 578:1,2,46 && 601:0,0,55 && ...". "578:1,2,46*" notation means that the term *"throughput"* appears in document which ID equals to 578 and it appears 1 time in title, 2 times in abstract and 46 times in body text. And finally indices are written to database.

C. Querying on resulted indices

Querying system enables querying on given keywords and returning list of documents ranked with relevance to the keywords. MySQL [10] will be used to hold indices obtained from indexing process.

We consider that if keyword appears in title, it should have more relevance than if it appears in abstract or body. Title has more weight than abstract, and abstract has more weight than body text. We will use one of the weighting schemes described in [5], called TF-IDF weighting.

5. Conclusion and Future Works

In this paper we described PDFindexer system which enables indexing and querying scientific articles in PDF. We described design of the system in detail and showed the interactions between the system components. Our main focus in this work is to provide scalable distributed indexing of PDF documents using MapReduce programming model.

Our implementation of the system is currently undergoing and in future works we will describe implementation issues and describe querying part in detail. Also we will conduct evaluation of our system testing suitability of indexing at larger scale.

References

[1] Google Scholar. http://scholar.google.com/ [cited in 2011].
[2] ScienceDirect. http://www.sciencedirect.com/ [cited in 2011].
[3] IEEE Xplore. Digital Library. http://ieeexplore.ieee.org/Xplore/guesthome.jsp [cited in 2011].
[4] J. Dean, S. Ghemawat. MapReduce: Simplified Data Processing on Large Clusters, OSDI'04: Sixth Symposium on Operating System Design and Implementation, Dec., 2004.
[5] C. D. Manning, P. Raghavan, H. Schütze. An Introduction to Information Retrieval, Cambridge University Press, 2008.
[6] T. Briscoe et al. Intelligent Information Access from Scientific Papers. The Information Retrieval Series, Vol. 29, Part 5, 2011.
[7] B. Goetz. The Lucene search engine: Powerful, exible, and free. Javaworld. http://www.javaworld.com/javaworld/jw-09-2000/jw-0915-lucene.html, 2002.
[8] Hadoop MapReduce. http://hadoop.apache.org/mapreduce[citedin2011].
[9] C. Ranger, R. Raghuraman, A. Penmetsa, G. Bradski, C. Kozyrakis. Evaluating MapReduce for Multi-core and Multiprocessor Systems. Proceedings of the 13th International Symposium on High Performance Computer Architecture.
[10] MySQL. Open-source RDBMS. http://www.mysql.com [cited in 2011].