JoySearcher

—A Multi-Keywords Based Retrieval System

With Hadoop

A small project for tasting cloud computing

1. Introduction

The hadoop was originally developed to support distribution for the Nutch search engine project [1]. In the original paper of MapReduce, Dean& Ghemawat [2] also presents a short description for performing indexing with MapReduce[3]. So we can say that the information retrieval (IR) promotes the birth of Hadoop. However, how to efficiently build the index with MapReduce still is a challenge problem. How to use the MapReduce to support the web search engine still is one of the top secrets of *Google.* Recent years, several open source project about building index with MapReduce have been launched, such as Katta, Bailey, Distributed Lucene, etc [4]. The IR in hadoop cloud system environment also is a hot area recent year, and some research results have been published [5].

In this report, I will describe my small project—JoySearcher. How to efficiently build the index is an academic problem. I do not have the ambition to settle this problem. However, I focus on building an available keywords based IR system—it can do something, even it cannot do it very well. So I call my project JoySearcher. In order to build JoySearcher, several components are necessary, for example, we need a crawler to crawl the web page from the WWW, a method to clean the data, a framework to build the index, etc. My work is focused on the later two parts. To build crawler also is a challenge problem. But I use an existing tool [6] to crawl the web page[+].

First of all, in order to build JoySearcher, I write a package to build index. This package has two functions; one is to clean the data (removing the tag of the data) which is crawled from the WWW. We can call it JoyCleaner. The other one is to build the inverted list of the keyword. We can call it JoyIndexer. In order to build the index, I design a data structure (InvertedList) to store the index. Inverted List support the index entity arranged in two orders (according the keywords or according to the ranking priority). It provides good support for the JoySearcher. I also try to write the program as the stand hadoop program, so it can be reused by others.

Secondly, I write another page to support the keyword based search. This part also can be called JoySearcher. JoySearcher can support keyword based IR. It has two functions. First, JoySearcher support multi-keywords based information retrieval. I present a mathematic model to support the multi-keywords IR based on the relations of the keywords. Second, the JoySearcher can support fuzzy query and precision query. It means the users can search the documents based on part of one keyword.

Thirdly, in the process of building project, I have an idea about the data compression for building index. However, because the limitation of time, I cannot do it. I will give a brief discussion about this idea.

To summarize, I make the following contributions:

1) This report introduces a whole processing of building a simple, primary keywords based information retrieval tools—JoySearcher.
2) In the report, I describe a simple mathematic model to support the multi-keywords based information retrieval. After analyzing the property of multi-keywords information retrieval, I set up a simple mathematic model to compute the ranking priority of different documents.
3) In the report, I design a data structure for building index of the document. The data structure is flexible and universal. This program of JoyIndexer also is a standard program, which can be easily reused by others
4) JoySearcher can support fuzzy query and precision query.

2. Key ideas and challenge

In this section, I will briefly introduce several main ideas and challenges of my project.

2.1 Multi-keywords Ranking Model

Given several keywords, we always need to fine the most related document. This just is the problem of JoySearcher.

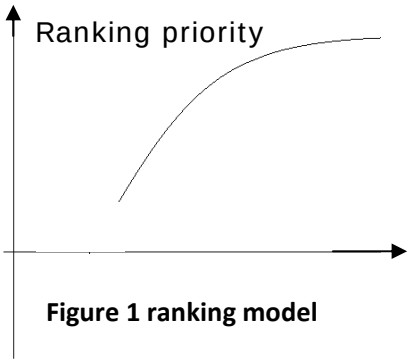

Figure 1 ranking model

It is easy to think that we can use the frequency of the keywords in a certain document as the ranking priority. I just select this as the basic ranking priority. However, we sometimes (or usually) compute the ranking priority of documents, which contain more than one keyword.

A simple method is to use the sum of the keywords' frequency. However, it is always not true. If a document only has two words, and both of them are query keywords, there is no doubt that this document is very important. But their sum may still be fewer than other documents. The other method is to use the product of their frequency. I think it is still unfair, because the priority of a document will increase exponentially according to the keywords. If a document only contains one kind of keywords, but the frequency of this keywords is very high, this document also should have a higher ranking priority. However, the product of their keywords frequency will be small. In my opinion, the ranking priority of a document should increase according to the number of keywords. However, their marginal should decrease according to the number of keywords, just as showed in the Figure 1.

As we can see from Figure 1, this curve is very similar with Logarithmic Functions. So JoySeacher's mathematic model of ranking priority is as following:

If there are a set of documents $d_i(0 < i < n)$, a collection of keywords $K_j(0 < j < m)$. Let D_i be the set of the words in document d_i, S_j be the frequency of K_j in a specific document d_i, c_i represents the amount of kinds of keywords in the document d_i. So the ranking priority of document d_i can be computed by

$$Rank_i = \sum_{K_j \in D_i} S_j \bullet \ln(1 + c_i) \qquad \textbf{Equation (1.1)}$$

Please pay attention that, if $c_i = 1$, the ranking priority of document d_i just is the frequency of the keyword.

2.2 InvertedList

I design an inverted list to store the index. This index is a universal inverted list. Just as showed in Figure2.

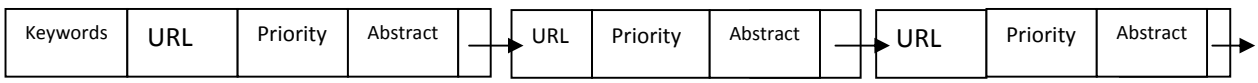

Figure 2 Inverted List

The main disadvantages of JoySeacher's Inverted List are in two aspects. The first one is all fields of the node in inverted list are implemented with generic class. So in different cases, the inverted list can be

implemented with different data types. This character guarantees that JoySearcher's Inverted List is universal.

The second one is that the Inverted List can be arranged according to two orders—the decreased order of the URL fields or the decreased order of the priority. It brings a great flexibility and convenience in multi-keywords based information retrieval. For multi-keywords information retrieval, we always need to merge several inverted list with same keywords in Reduce Phase. Traditional inverted list is ordered by priority, so the cost of merging is high. However, in JoySearcher, the inverted list is ordered by URL. So in the Reduce Phase, we can merge them more quickly with k-run merging method. As showed in Figure 3, if we want to merge three list into one, every time we just select the large node from the head of each list. If more than one node has the same key, we merge them into one node. In figure 3, there will be only one node with key"H" in the merged inverted list Inverted List also can be resorted according to the priority and return to the users.

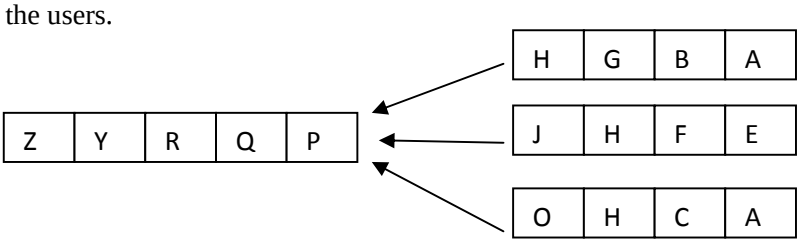

Figure 3 k-run merging

2.4 Two query paradigm

The JoySearcher supports two query paradigms. One is precision query, the other one is fuzzy query. For the precision query, it is easy to understand. The JoySearcher just look up each keyword with precision detection. For the fuzzy query, the JoySearcher will treat each keyword as the regular expression, and detect them according to formal grammars. For example, for precision query, the COM and COMPUTER are two keywords. However, for fuzzy query, the COMPUTER can be represented as COM*. They are same keyword for fuzzy query. Java provides wrapper class for regular testing. So the two query paradigm can be easily implemented.

3 Components of JoySearcher

I have given a brief description of the architecture of JoySearcher. Now I will give a more extensive discussion about the different components of JoySearcher.

As I have mentioned, the JoySearcher has four components: JoyCrawler, JoyCleaner, JoyIndexer and JoySearcher (this is also si JoySearcher, but it will not cause confusion according to the context).The data stream will go through them one by one. Figure 4 shows the data flow of JoySearcher.

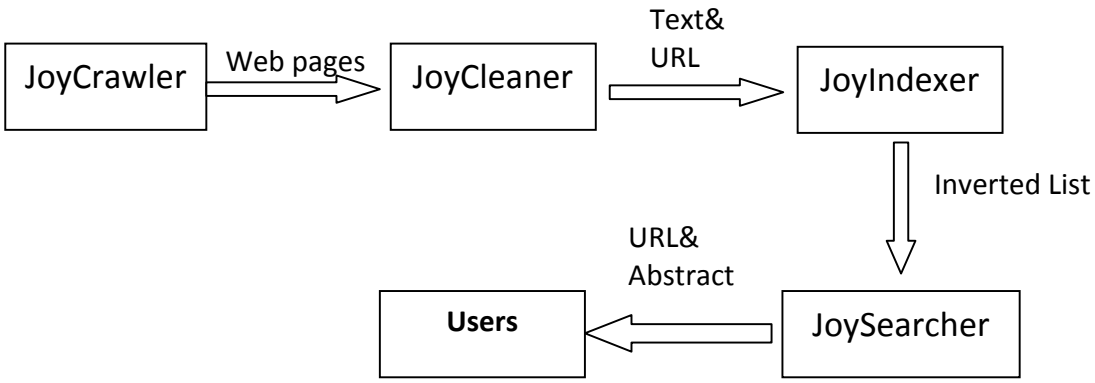

Figure 4 the data flow of JoySearcher

3.1 JoyCrawler

The first component is JoyCrawler. JoyCrawler is complicated; it even is more complicated than sum of the other parts of JoySearcher. In fact, the whole JoyCrawler is not my work. It belongs to Song Liu [6]. JoyCrawler is a small open source project. It is not as complicated as the Nutch, but has the similar mechanism of Nutch. It also is a standard hadoop program. In my project, the JoyCrawler ‘s version is JoyCrawler-0.11.1, now it has been JoyCrawler-0.2

3.2 JoyCleaner

The second component is JoyCleaner. For JoyCleaner, I use DOM to parsing the web page. The XML DOM defines a standard way for accessing and manipulating XML documents. The DOM presents an XML document as a tree-structure [8]. There are other methods to parse the web page, for example, we can analyze the tag of HTML. However, those methods are slower and more complicated. I should highlight it that JoyCleaner and JoyIndexer is put into a same package, because their task is the same—to construct the index.

3.3 JoyIndexer

The third component is JoyIndexer. I spent a lot of time on JoyIndexer. I try my best to write the JoyIndexer as a standard program and can be reuse by others easily. The program of JoyIndexer sticks up to Open-Close Principle (Software entities should be open for extension, but closed for modification [9][10]). There are three main kinds of class; they all use the composition association, and avoid extends.

Figure 5 shows the basic UML of this package.

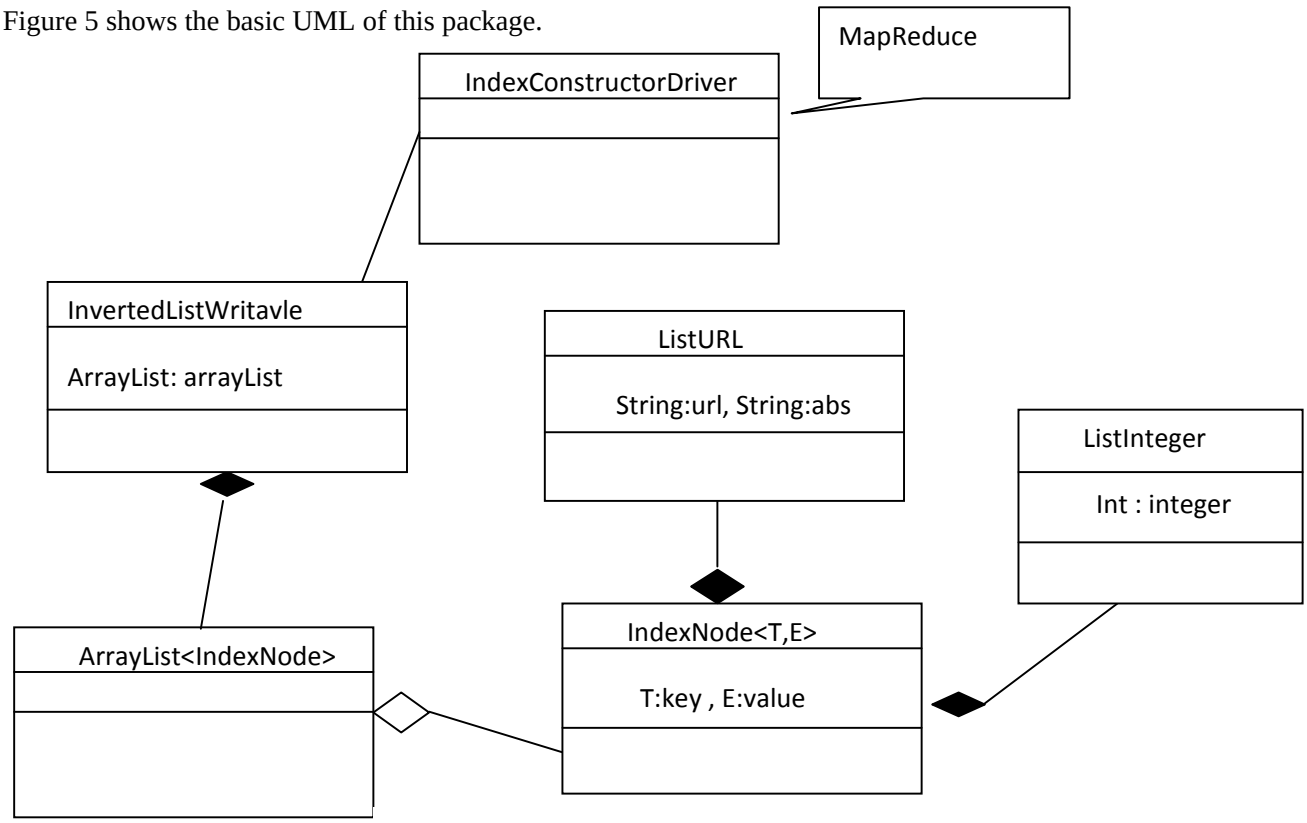

Figure 5: The UML of JoyIndexer

3.4 JoySearcher

JoySearcher is the function module of the system. I have talked about the JoySearcher extensively in section 2. Firstly, the JoySearcher support multi-keywords retrieval. If there are multi-keywords query, the JoySearcher can merge the multi-keywords InvertedList into one InvertedList. The InvertedList provide enough support to make the merging efficiently. During the merging process, the ranking priority of documents can be re-computed by the Ranking model in section 2.1.

Secondly, the JoySearcher support two query paradigm— fuzzy query and precision query. I utilize java wrapper class to implement fuzzy query paradigm.

5. Acknowledgements

My special thanks must go to Song Liu. The original idea of JoySearcher just comes from JoyCrawler.

6. Reference

[1] Hadoop;http://en.wikipedia.org/wiki/Hadoop;

[2] Jeffrey Dean, Sanjay Ghemawat: MapReduce: Simplified Data Processing on Large Clusters. OSDI 2004: 137-150

[3] Richard M. C. McCreadie, Craig Macdonald, Iadh Ounis: On single-pass indexing with MapReduce. SIGIR 2009: 742-743

[4] Mark H.Bulter, James Rutherford: technical report : Distributed Lucene: A distributed free text index for hadoop, HP lab, June 7, 2008

[5] Sai Wu, Kun-Lung Wu: An Indexing Framework for Efficient Retrieval on the Cloud. IEEE Data Eng. Bull. 32(1): 75-82 (2009)

[6] Song Liu, JoyCrawler, http://joycrawler.googlecode.com/ , 2009,11

[7] Richard M. C. McCreadie, Craig Macdonald, Iadh Ounis: Comparing Distributed Indexing : To MapReduce or Not? ,LSDS-IR workshop. July 2009, Boston, USA.

[8] XML DOM Tutorial ; http://www.w3schools.com/dom/default.asp, 2009

[9] Open/closed principle, http://en.wikipedia.org/wiki/Open/closed_principle; 2009

[10] "The Open-Closed Principle", C++ Report, January 1996