Project 4: Putting It All Together
CS453 Web Data Management

Hoang Le Huy Tran Bo Wang

December 12, 2012

1 Introduction

1.1 Program Overview

In this project, we combine everything we have gotten from the previous projects. The major task of this project is to process the query over multiple EC2 nodes and to design the user interface. Our program fulfills all the requirements from the project description [3].

1.2 Roles of Our Team Members

1.2.1 Hoang Le

He is responsible for completing the following tasks.

- Design and implement distributed search engine architecture
- Implement TDIDF and BM25 algorithm
- Design method combining pagerank and relevant scores
- Implement back-end supporting for suggestion search
- Implement voice search feature

1.2.2 Bo Wang

He is responsible for completing the following tasks.

- Implement search algorithm in slave node
- Implement AND OR query processor
- Implement the Query Likelyhood model
- Complete the "Did you mean" feature

1.2.3 Huy Tran

He is responsible for completing the following tasks.

- Design the system architecture
- Design the web application
- Design method combining pagerank and relevant scores
- Complete instant search feature
- Implement front-end supporting for suggestion search

1.2.4 Team Work

We met and discussed regularly about the design of our search engine architecture. We also communicated via emails for updating the tasks' status of each team member. The major topics we have discussed are

- How do we distribute the inverted list efficiently?
- How do we display the content of the web application?
- We evaluate different solutions and select the best solution.

Besides, we used Dropbox [4] and Google Drive [5] for sharing our source codes and documentation. In this project, we leveraged storage provided by Amazon S3 [2] to store all of our needed data.

2 System Design Background

In this section, we describe how we build each part and how they work together.

2.1 System Overview

The figure 1 shows our system design.

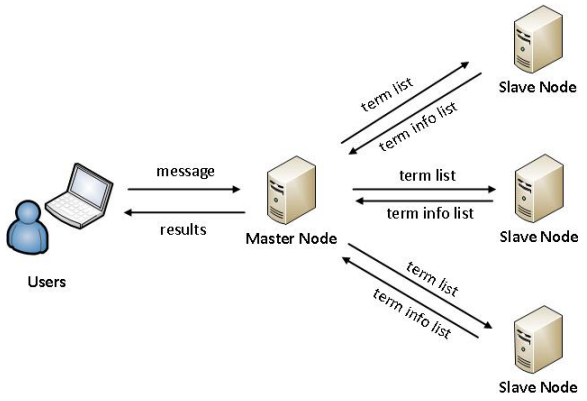

Figure 1: System design

We describes the data exchanged in our system below.

- Message contains the query typed by an user, the query type, which is AND or OR query, the model the user wants to apply, and parameters corresponding to each model.
- Results contains information to be display in the web application.
- Term List contains list of terms.
- Term Info List contains terms and their associated information.

We use Hypertext Transfer Protocol (HTTP) to exchange the message and results between the master node and the web application. Term List, and Term Info List are exchanged by using Transmission Control Protocol (TCP).

The following sections, we will describe in detail how our system works.

2.2 Distributed Query Processing

In this section, we describe our distributed search engine architecture.

The purpose of this architecture is to enable high amount of fast query processing on our system, and also to adapt to the limitation of realistic web data collection over network connection and machine power.

Our system makes use of four nodes over Amazon EC2. One node (the master node) will deal exclusively with the web user interface. The inverted list are spitted among other three slave nodes in a balancing way of both storage and amount of requests.

When the master node receives a query $Q = (t1, ..., tk)$, the master will request the k lists from their corresponding slaves. There are three sub-requests are sent to three slave nodes. Each sub-request is guaranteed to be correspondent to the content, which is portion of inverted list, stored in each slave node.

Below, we describe our distributing algorithm:

1. Sort the inverted list in lexicographical order of the term
2. Split the sorted list into three continuing sorted sub-lists so that every sub-list has the same sum value of all terms frequency inside it. For each sub-list, we keep the first and the last term.
3. Distribute each sub-list to each slave node

This method works because it makes sure that the probability of requests sent to every slave nodes is the same because the total term frequency in all nodes are equal. Also, the total term frequency in the collection is proportional to the term frequency from users' queries.

Moreover, the frequency of a term is roughly similar to the probability that a term appear in many documents, which affect to the size of inverted list. This means that our distributing method also maintains the size of portions of inverted list in every slave node are roughly equal.

In addition, by storing the first and last term in each sub-list in the master node, it allows the master

node to split the query and send the correct request to a correspondent slave node.

2.3 Retrieval Model

The retrieval model can be divided into three parts.

- In the first part, the master node creates a thread to handle each query from user and parses it into terms. Then, the thread chooses which slave node to send each term according to the lexicographic order of that term. After this, every slave node creates a thread for each term they receive and search from the distributed inverted list. The slaves will return a document list that contains that term to master.
- The second part is to process the documents with 'AND' or 'OR' operation according to user's selection. This will return a filtered list of documents with related information.
- In the third part, we calculate the score for each document in the list with different models (TFIDF, BM25 and Query Likelyhood, yes we did all of them). We then sort the result and return ten items to user based on which page one chooses, e.g. 1-10 for page 1, 11-20 for page 2.

2.3.1 Combination of Pagerank and Relevance Score

Since the outputs from TFIDF, and BM25 models are only relevance scores which represent how relevant documents to the search terms, we need to combine these scores with the page ranks which represent how influences of pages. After combining these values, we have the final score which is the final metric.

$finalscore = C * norm(\text{relevant score}) + (1 - C) * norm(log(\text{page rank}))$

where

$norm(x) = (x - min(x))/(max(x) - min(x))$
with x is a relevant score or page rank.

C is the input from user $0 \leq C \leq 1$

In this formula, we use $log(pagerank)$ because one of the results from the project 3 [**?**] showed that the page ranks of 90 percent of documents in our collection are less than 1 while page ranks of 10 percent of the documents are very high. It means that there is an exponential increasing in the distribution of page ranks. Therefore, we $log(pagerank)$ is to minimize the effect caused by this characteristic.

Besides, we normalize the relevant score and the $log(pagerank)$ to 1.

Finally, by changing the value of C, users can customize their search by considering which score is more important to them. By default, we set $C = 0.5$.

2.3.2 Query Likelyhood

The query likelyhood model evaluates the document rank score by balancing the contribution of document being evaluated and all the background documents. We use parameter Lambda to control the balancing. As Lambda getting smaller, the background documents becoming less influential. In the extreme case, if Lamba equals 1, we only consider the background documents' contribution, which is almost equal among every document, so the dominating factor is the original page rank. We therefore set the default Lambda as 0.1 to lay more emphasis on the document relevance.

2.3.3 Compare between Different Models

After 20 different search queries with TFIDF, and BM25 models with default configuration, we see that BM25 provides more relevant results to the users. We did not compare Query Likelyhood model with other models, but Query Likelyhood provides relevant results to users.

2.4 Interface

The figure 2 shows the interface of our web application. We display all required information described in project 4 [3].

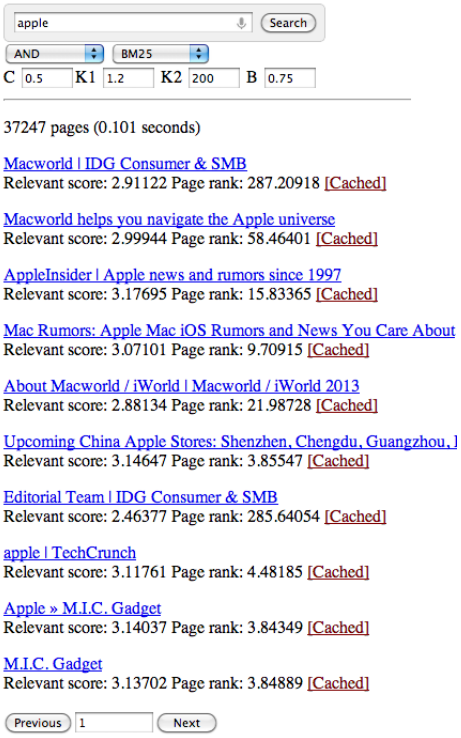

Figure 2: Web application interface.

2.5 Extra Features

We attempted all of the extra items in this project. Here is a brief introduction about how we do each of them.

2.5.1 Instant Search

On every keystroke in the query interface, the web application sends a HTTP request to the server. After receiving results, it updates different parts of the web interface immediately. We follow exactly AJAX [1] which is the art of exchanging data with a server, and updating parts of a web page.

2.5.2 Suggestions

There are two three steps in completing suggestion feature.

- First of all, we get a list of popular terms based on the term frequency after removing terms in the stop list.
- Second, for each term in the list above, we scan through our collection to find a term which often follows it. After this step, we have a list of popular terms with a term which often follow it.
- At the beginning, the web application requests for the list above and store it locally. The when the user types characters, the feature auto complete provided by JQuery [6] library will be triggered and display a list of suggestions.

2.5.3 Did you mean

We found that the Google did you mean algorithm works like this [8]: If the users perform a query, and only 10% of the users click on a result and 90% goes back and type another query (with the corrected word) and this time that 90% clicks on a result, then they know they have found a correction. However, we don't have the resource that Google has, so we use another approach which is based on a probability algorithm [7]: First we load a big text file, and count the frequency of every word and make a frequency table based on it. For each term in the query, we perform the following operations: insert, delete, replace, transpose. By insert, we can insert an character to any place of the original word; by delete, we can delete any character of a word; by replace, we can replace an character with any other character; and by transpose, we can switch the position of any two adjacent characters. Any one of the above operations is called an edit of distance 1. The literature on spelling correction claims that 80 to 95% of spelling errors are an edit distance of 1 from the target. The next step is to find all the candidates that are distance one from the mis-spelled word. Then among these candidates, we choose the one with highest frequency in the text file to be the correction.

2.5.4 Voice Search

We employ Voice search tool in HTML5 in our system. This tool is supported by Google. This tool con-

tains a speech recognition module running directly in the client side, which is the Google Chrome Browser. This module recognize English speech and return text result input to the search box.

3 Challenges and Conclusion

3.1 Challenges

Working on this project, we have had to face with several challenges. The requirement of this project is open, so we need to find the best possible solution. Also, deploying the system on Amazon Cloud is not a trivial solution. In addition, web programming is not one of our familiar skills.

3.2 Conclusion

In this final project, we have developed a complete searching system based on our web page collection and on all information that we have extracted in previous projects. Our solution not only fulfills all the major requirements for this project, but also includes all the extra features listed in the bonus section. To the best of our knowledge learnt from this class, we have tried and employed all possible architectures and algorithms to optimize our solution for this project, for example : distributed search engine architecture, page ranking models and retrieval models, topical relevance algorithm : TFIDF, BM25, Query Likelihood ... Moreover, by finishing the complete system, we have an opportunity to learn and practice many other skills, such as developing algorithm for distribution model, network programming, TCP/IP, server-client model, and also some of web programming. Finally, we have learned and practiced how to build a complete web searching system.

4 Reflections

4.1 Course-Based Questions

4.1.1 Hoang Le

1. The most useful thing I have learned in this class is how to develop a web-searching system employed some of information extraction algorithm, and especially the skill how to handle big amount of data. Also, Im interested in map-reduce algorithm.

2. I have got valuable experience handling big amount of data. I have known handling big data is not easy and a good algorithm plays an important role on big data.

3. I have learned map-reduce which is an impressive algorithm for handling big data. I was also practicing a lot with parallel programming.

4. Through experimental analysis, I think that our system works pretty well. We have tried our best to develop the core of our searching engine and have seen what we expected from it.

5. If there was more time, I would have like to learn more about map-reduce algorithm. To make more time, I would have preferred to see less about XQuery, XPath.

4.1.2 Huy Tran

The most useful thing I learned in this class is applying map-reduce paradigm in analyzing large data set. The course shows me the importance of managing the big data in the real problem. It is especially useful that I learned how to write effective parallel and distributed code. Also, by doing the evaluation part of reports, I can improve my critical thinking about the inner workings of our system. If I have more time, I would have liked to learn more about the distributed system. To make more time, I would have preferred to see less about XML and XQuery .

4.1.3 Bo Wang

1. The most useful thing I learned in this class is the experience to work with a team and finish a big project step by step. I learned how to discuss a problem, distribute the tasks, and cooperate with other teammates efficiently. Also I get a good understanding of the course material by doing different projects. Besides, my programming skills gets improved a lot.

2. This course gives me a very good introduction and understanding of the big data management in practice. (b) It shows lots of examples of parallel/distributed programming architectures in real world, and I got a chance to write some of them. (c) Sometimes details in realizing the system are not covered in the lecture, this requires us to think critically, which help me to understand things better. We also have the flexibility to implement our own thought.

3. If there was more time, I would have liked to learn more about noise reduction, encoding and language analyzing.

4. To make more time, I would have preferred to see less about retrieval models.

4.2 Project-Based Questions

4.2.1 Hoang Le

1. What I enjoyed most is working on project, applying theory from class to solve the real problem.

2. What I enjoyed least is exams and all paper questions in the first part of the class.

3. Ive enjoyed writing the project report, except the first one, we have not been well-prepared for it. Yes, because of the demo, I have to work closely with my team and have to understand our work clearly. It keeps me focus on our project and understand the problem and our own solution better.

4.2.2 Huy Tran

1. In general, the thing I enjoy the most is applying the knowledge that I learned in the class into these projects.

2. The shared writing, experiments, and analysis in our project reports are valuable for a graduate student like me.

3. The project demos improve my learning because I can ask the instructor some questions which I could not write in the report. I can also explain more clearly with the instructor my ideas.

4.2.3 Bo Wang

1. I enjoyed Project4 most. Because it combines everything, I had a chance to learn different skills and get a better understanding of how the whole system works. I enjoyed project 2 least, because there is not much to do and the instruction is kind of vague.

2. Yes, writing and analysis are helpful and valuable exercise to me.

3. Yes, the advisor always have unexpected questions during the demos, which helps us think of questions that we havent noticed or cared too much before. It makes me think critically.

References

[1] AJAX. Ajax http://www.w3schools.com/ajax/default.asp.

[2] AMAZON. S3 http://aws.amazon.com/s3/.

[3] D. Chiu. Project4-cs453 http://encs.vancouver.wsu.edu/ chiu/cs454/project4-f2012.pdf.

[4] Dropbox. Dropbox https://www.dropbox.com/, October 2012.

[5] Googledrive. Googledrive https://drive.google.com/, October 2012.

[6] JQUERY. Jquery http://jquery.com/.

[7] P. Norvig. How to write a spelling corrector http://www.norvig.com/spell-correct.html.

[8] Stackoverflow. machine learning - how does the google did you mean algorithm work? http://stackoverflow.com/questions/307291/how-does-the-google-did-you-mean-algorithm-work, December 2010.