AskUbuntu - Data Analytics

Andre Cunha*
School of Engineering and Applied Science
Columbia University
*adc2171@columbia.edu**

***Abstract*—Ask Ubuntu is a question and answer site for Ubuntu users and developers. It is built and run by many users as part of the Stack Exchange network of Q&A sites. AskUbuntu is well known for provide an easy and free platform where good answers are voted up and easy to find. The main objective of this project is to improve AskUbuntu service by developing a feedback/recommendation system where suggestions of questions or comments will be offered to the users every time they are writing specific key-words.**

***Keywords*-Big data; Spark MapReduce; Python; Tag-it; JavaScript; PHP; MySQL; Apache;**

I. Introduction

AskUbuntu is a question and answer site for students, professionals, and researchers that aims to make technological information easily accessible to everyone. It has a huge dataset containing millions of posts and comments about Ubuntu operational system, including its softwares and applications.

AskUbuntu offers a unique approach to different users. Its dataset is well known to provide students comprehensive subject coverage without the information overload of a general search engine.

I decided to use this database because of its academic value, and I hope to contribute for AskUbuntu mission, which is to share and grow the world's knowledge.

That being said, I aim to optimize AskUbuntu web platform by analyzing and improving its public available dataset. Based on specific key-words and tags, I will make a final feedback/recommendation systems containing the most relevant questions related to a given matter.

II. System Overview

One of the common problems in data science is gathering data from various sources in a somehow cleaned (semi-structured) format and combining metric from various sources for making higher level analysis. For this project, I used a public avaiable dataset provided by StackExchange [1]. It is a 5 GB file with eight XML files. Table I highlights these files and their structure.

Our system consists mainly of two parts. The first one is a data processing $backend$, and the second is a modeling and analysis $frontend$ application. Figure 1 shows the system overview.

[1]https://data.stackexchange.com/

Table I
Dataset Structure

Badges.xml	Stores Users' Badges (Student, Supporter,...)
Comments.xml	Stores Users' Comments (PostId, Score, Text,...)
PostHistory.xml	Contains the Posts Edit History
PostLinks.xml	Holds the relationship between posts
Posts.xml	Contains the Posts and their information
Tags.xml	Stores the Key-Word/Tags lists
Users.xml	Contains Users' related information
Votes.xml	Keeps track of the vote system

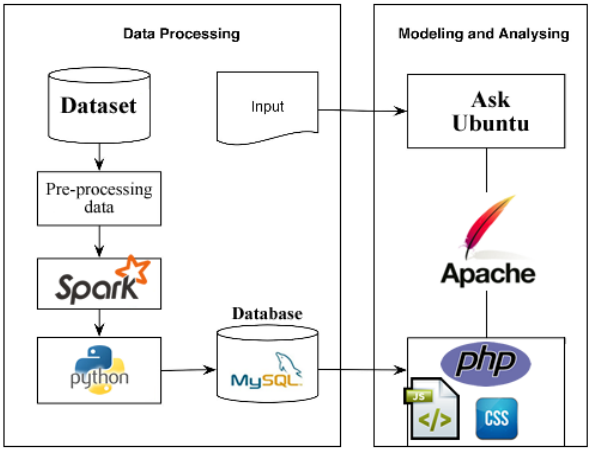

Figure 1. System Overview.

A. Backend

Our backend structure was built upon a Spark cluster application. Apache Spark is a fast and general-purpose cluster computing system. It provides high-level APIs in Java, Scala and Python, and an optimized engine that supports general execution graphs.

At a high level, every Spark application consists of a driver program that runs the user's main function and executes various parallel operations on a cluster. The main abstraction Spark provides is a resilient distributed dataset (RDD), which is a collection of elements partitioned across the nodes of the cluster that can be operated on in parallel. The Spark Python API (PySpark) exposes the Spark programming model to Python.

This project was built using Python, which is dynamically typed. That means, all the collections of RDDs can hold

objects of multiple types. In PySpark, RDDs support the same methods as their Scala counterparts but take Python functions and return Python collection types.

Apache Spark is a highly fault tolerant system and it is designed to be deployed on low cost hardware. It provides high throughput access to data and hence is suitable for applications with large datasets. The softwares used were:

- Ubuntu 14.04
- Spark 1.3.1
- Scala 2.10.4
- Java 1.7
- Python 2.7

Spark was responsible for perform a Task Tracker running a MapReduce processing layer and a DataNode in my local storage layer. Spark Core API coordinated and managed the tasks and processes using the following stack diagram.

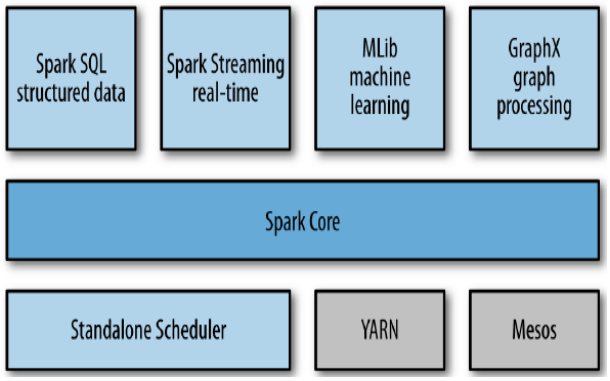

Figure 2. Spark Stack Diagram.

PySpark allows users to write map/reduce code in Python using the Spark Core API. This is a key feature as it means creating a Python script into a Spark does not require learning a new language like Java or derived Hadoop-centric languages (i.e Pig). Based on different guides, I developed a Python code to perform a MapReduce algorithm.

The mapper algorithm is responsible to take the raw text of AskUbuntu XML dataset, organize the data, and convert it to key/value pairs.

Each key corresponds to a specific key-word/tag available at $Tags.xml$ file. The value is a concatenation between the fields: *Post ID, Post Rate Metric, Post Title*. The algorithm provides the most relevant Posts for each key. Table II illustrates how the (Key, Value) scenario was configured.

Table II
MAPPER OUTPUT SAMPLE

Key-Word [*Post ID, Post Rate Metric, Post Title*]
('ruby', [[u'95', 89.68, u'Completely Remove Ruby + Rails + Gems?']])
('pointers', [[u'116', 80.502, u'How do I install more pointers?']])
('debian', [[u'167', 23.08, u'How do I use debain repositories?']])

The reduce step will group all keys, and create a list with their respective values. This will reduce the output to a list of unique keys, each with a value corresponding to different questions.

B. Frontend

The first step to build our frontend application was to set up a LAMP server into our desktop computer. According to wikipedia, LAMP is an acronym for an archetypal model of web service solution stacks, originally consisting of largely interchangeable components: Linux, the Apache HTTP Server, the MySQL relational database management system, and the PHP programming language.

As a solution stack, LAMP was a perfect match for our project. It is suitable for building dynamic web sites and extremely easy to integrate with our backend application.

The main core of my frontend application is Jquery Tag-it[2]. Jquery Tag-it is an APIs that support users incorporate robust functionalities into their own website and applications. Taking into consideration our project, we chose it because it is a simple and configurable tag editing widget with autocomplete support.

Tag-it was integrated Google with a LAMP server using JavaScript/PHP to perform SQL queries into a MySQL database. Remember that our database stores a list of unique keys and values generated by our backend application.

A PHP script was designed to easily access these values and perform actions based on user input parameters. Our frontend application is finalized with a user interface created by CSS and HTML.

III. ALGORITHMS

In this chapter, the two methods to achieve my final goal will be introduced. I researched several tools and algorithms for the purpose of processing and XML files. However, any of them fitted my needs.

Therefore, I decided to use Python Regular Expression to parser XML files and develop my MapReduce approach. I found out that advanced and robust applications can be developed in a Python+Spark environment, and then, applied to a frontend service with a friendly user interface. We have made an early decision to focus on Python and a LAMP application due to three main reasons.

The first, and most important one, is that I am already familiarized with these technologies. Second, Spark Core API turned out to be an useful tool to create a Map/Reduce job, submit the job to an appropriate cluster, and monitor the progress of the job until it completes. Third, Jquery Tag-it API provides useful information easily accessible throughout JavaScript.

[2]http://aehlke.github.io/tag-it/

A. *Python & Spark = PySpark*

Our Python algorithm was based on a MapReduce approach to organize and manipulate AskUbuntu dataset. Spark was responsible to administrate the parallel tasks and processes. A mapper task will execute a Python script as a separate process when initialized. As the mapper task runs, it converts the data originally from XML files and feeds the reducer process, which organizes the data into a key/value pair. My *Mapper Algorithm* was implemented at the file $AskUbuntu.py$, and it follows the pseudo-algorithm:

1) Read lines from a XML input file
2) Remove leading and trailing whitespace from input data
3) Use Regular Expression Functions to Read/Parse the dataset
4) Create a list of all known key-words from the file Tags.xml
5) Calculate the Rate Metric Parameter
6) Process the file Posts.xml creating lists [Key-Word [*Post ID, Post Rate Metric, Post Title*]]
7) Output the result

A reducer task is responsible to group all keys created by the mapper process. Its class was created at the file $AskUbuntu.py$ and obeys the following pseudo-algorithm:

1) Receive input data from the Mapper Process
2) Group (spark.groupByKey()) the keys based on unique values of Key-Words
3) For each group:
   - Evaluate the Questions/Comments
   - Select the Top 3 of them
4) Output the result

Figure 3 illustrates the mapper and reducer tasks.

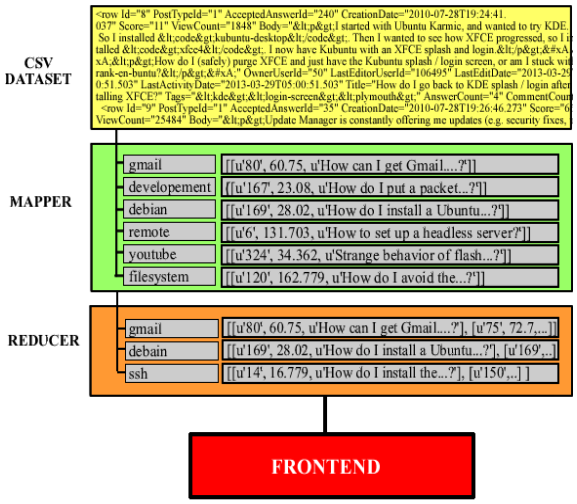

Figure 3. MapReduce Approach.

To evaluate different questions and comments, I decided to calculate their performance using six terms. Figure 4 summarizes all of them.

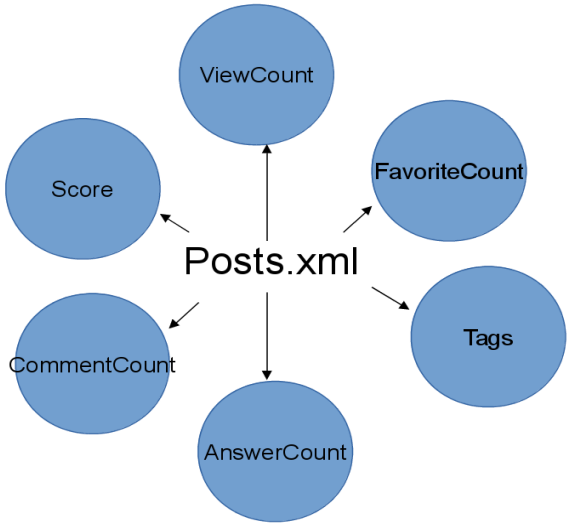

Figure 4. Rate Metric Terms.

Based on the previous information, my algorithm must rank all the questions and comments from the file Posts.xml using a rate metric.

To calculate the rate metric, my algorithm assigns different weights in function of the six terms detailed at 4. Table III highlights these weights and equation (1) defines my final rate value.

Table III
WEIGHTED TERMS

Weight	Parameter
1.5	Score
$1/1000$	ViewCount
3	AnswerCount
2.3	CommentCount
3.3	FavoriteCount

$$Value = (Score) * 1.5 + \frac{ViewCount}{1000} + (AnswerCount) * 3 + (CommentCount) * 2.3 + (FavoriteCount) * 3.3 \quad (1)$$

B. *Jquery Tag-it, PHP & MySQL*

Jquery Tag-it is an APIs that support users incorporate robust functionalities into their own website and applications. Jquery Tag-it outputs a list of all key-words a user is writing. Figure 5 highlights how I integrated that list with a LAMP server using JavaScript/PHP to perform SQL queries into a MySQL database.

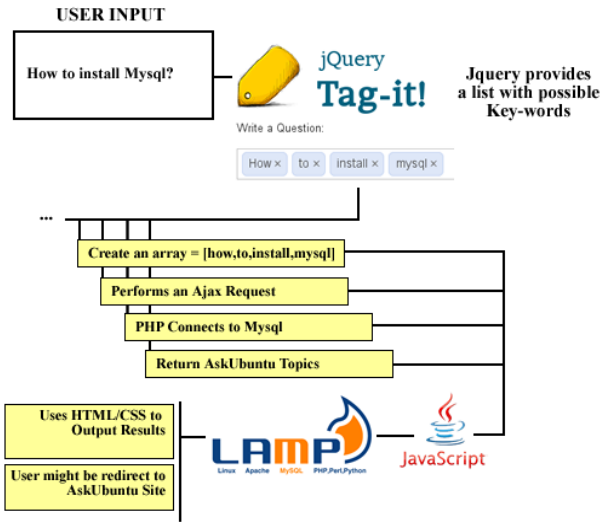

Figure 5. Jquery Tag-it + LAMP server.

IV. Experiment Results

After develop and test my project, I am extremely happy about my final results. I have learn how to use Spark as a dynamic tool to solve real-worlds problems, and applied different techniques with Python scripts and a LAMP server.

My Map/Reduce algorithm demonstrated a high level of controllability and performance. To verify my results, I tested my frontend application for different scenarios. The figures below show a few questions that I had in my mind and used to verify the credibility of my algorithm.

Figure 6. Question 1.

Figure 7. Question 2.

As described in the figure above, for the questions 2 and 4, my rate metrics were of 260 and 372, which are a terrific results. It means that both suggestions have a high probability to contain what I am really looking for. After clicked on the recommended link on the bottom, I was redirected to AskUbuntu web platform, and there I found the answer for my question.

Figure 8. Question 3.

Figure 9. Question 4.

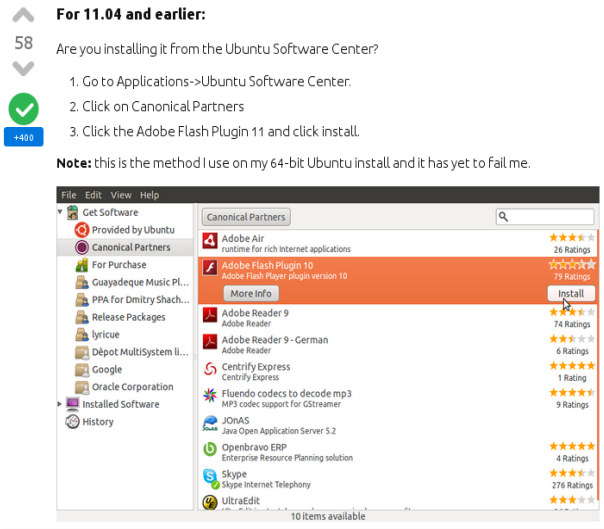

Figure 10. Answer for the question: How to install flash?

On the other hand, for the question 3, my algorithm performed poorly. I got a AskUbuntu post with a rate metric of 38 only. There might be different reasons for this. I assumed that nobody asked about it on AskUbuntu, or if someone commented about it, he used different key-words. That indicates a flaw on my code, because I am relaying exclusively on key-words to perform my recommendation system.

V. Conclusion

I implemented and described a practical solution to address a simple and easy-to-use recommendation system. I built my system analysis based on two different applications. The first one, a backend service, was constructed upon a

Spark Core API supported by a local file system. I presented a simple MapReduce program for PySpark written on Python programming language to rank different AskUbuntu questions and comments using a rate metric that I developed based of weighted key-words.

The second application was a frontend web-site where an user can easily visualize recommended topics on the AskUbuntu web platform that might be related to their key-words. This web-site was built using PHP, Mysql, JavaScript, and Jquery Tag-it to support our backend algorithm and create a friendly user environment.

Although the project comes to an end, I feel that my task can still be deeply extended. Firstly, I can not only provide the ranking information to users, but also give a more specific information based on my recommendation system. In addition to that, we can combine the information from other public datasets to give a overall prediction to users. All in all, I am looking forward to spread this project and make people's life more convenient.

REFERENCES

[1] Holden Karau, Andy Konwinski, Patrick Wendell, Matei Zaharia: Learning Spark: Lightning-Fast Big Data Analysis, 2014

[2] Kawa, A., Bolikowski, ., Czeczko, A., Dendek, P.J., Tkaczyk, D.: Data Model for Analysis of Scholarly Documents in the MapReduce Paradigm. In: Bembenik, R., Skonieczny, L., Rybiski, H., Kryszkiewicz, M., Niezgdka, M. (eds.) Intell. Tools for Building a Scientific Information. SCI, vol. 467, pp. 155170. Springer, Heidelberg (2013)

[3] Kimball R, Ross M. The data warehouse toolkit: The definitive guide to dimensional modeling[M]. John Wiley & Sons, 2013.

[4] [10] Van Rossum G, Drake Jr F L. Python tutorial[M]. Centrum voor Wiskunde en Informatica, 1995.

[5] McCallum, A.K., Nigam, K., Rennie, J.: Automating the construction of internet portals with machine learning. Information Retrieval, 127163 (2000)