Project Report: Guild – Talent Search Engine

By: Jivjot Singh, Bikramdeep Singh

1. Problem Definition:

If we went around and asked all the CEO's of the top technological companies and ask them what is the most challenging and difficult task in running the company, they will most probably answer that hiring and retaining the right talent is the hardest part of running a successful technological company.

In this project we focus on the hiring part of the problem. Over the last decade we made long strides in efficient talent searching and filtering out the best people. Now we have automated Resume and C.V parsers which automatically extract skills and keywords, which are used for candidate filtering. Same can also now be done using professional social network websites, such as LinkedIn, which provide more structured and clean form of data.

Although such tools are efficient, but the problem is that these results are based upon on self reported data, which may not be entirely accurate. Moreover, it is not numerically quantifiable, such as if 2 candidate reports advance skills in C++, how do we find who is better. We need a better way to authenticate such claims.

One possible way is using standardized, screening test, programming contest platform such as hackerrank.com is such an example. But not all skills can be tested in such a way, moreover such as process is costly and time consuming.

2. Methodology:

We suggest here to use StackOverflow data to find the right candidate. StackOverflow is a subgroup of StackExchange websites, which is an online Q&A platform. But where major focus is towards software engineering and technological frameworks. On this platform more than 4,000,000 users are registered, due their large community and technological specialty, it has become a kind of peer reviewed global talent hub. Each user can post questions, answer a question, up vote or downvote a response. Due their community driven approach a user can earn reputation points by participating in their Q&A forums. Moreover, it has a feature which allows users to tag different questions with different technology-domain.

If we are able to calculate a score or a reputation of different users based on different tags, we could provide hiring managers an efficient tool to search ideal candidate, where they can input different skills and the system could return candidates which fit the given subdomain. Moreover, the system will return a numerical score, so that different candidate can be compared.

We wish to create a simple User Interface for the hiring managers, where they would input different skills/tags and our system would return the top candidates for this particular skill set. So in short we need to analyze the dataset so we have a score of each user for the particular combination of tags.

2.1 Tools:

Spark Flask web development, one drop at a time python™

APACHE HBASE Apache Thrift ™ HTML 5

node JS™ ANGULARJS by Google CSS 3

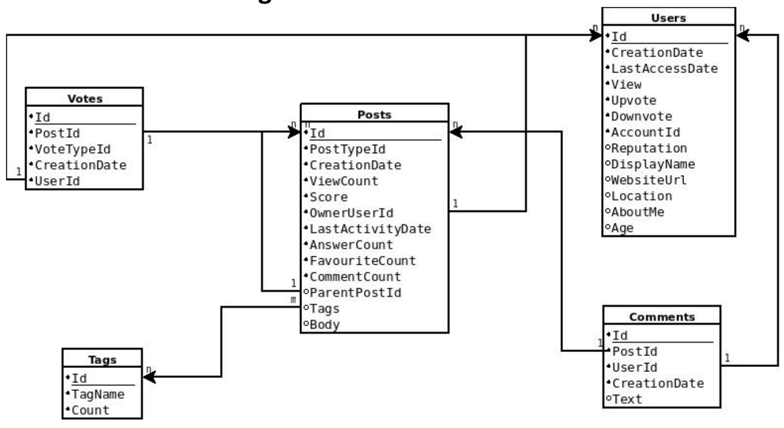
2.2 Dataset:

StackExchange.com, the parent website of StackOverflow provides their whole database dump which is available at archive.org. This data dump contains data of all the StackExchange websites. But we are only concerned about StackOverflow data, which in any case forms 90% of the total data.

Each StackExchange website dump contains compressed XML files representing tables in their database. Important XML files are described below.

- **Users.xml:** This file contains user profile information, such as Name, Title, Age, Location
- **Tags.xml:** This file contains all the tags used in the website to mark questions. As tags are created by users, it is an ever growing list.
- **Posts.xml:** This file contains all the questions and their corresponding answers. Questions and answers are separate entry here, where they are differentiated by attribute 'PostTypeId'. This table is linked to single user (the writer of the post) and multiple tags
- **Votes.xml:** This file contains all the votes recorded on 'StackOverflow'. 'VoteTypeId' attribute lets us differentiate between 'UpVote', 'DownVote', 'Accepted', 'Spam' etc

Figure 1: Detailed Schema



Note: Some tables and attributes are omitted for clarity

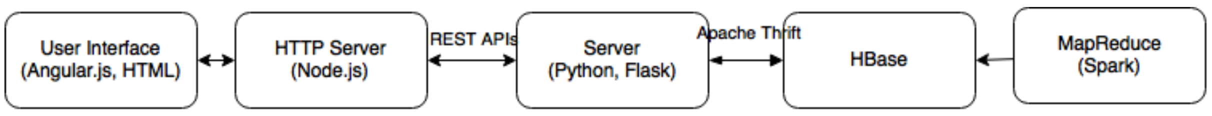
2.3 Data Preprocessing:

In order to parse XML in spark, we tried package available here but were not successful. Uncompressed xml files takes about 25 GB of space, so in order to parse it, whole thing need to be memory. Instead of parsing the whole file we processed the file line by line using spark jobs. Each row information was contained in row tag, which only spanned in single line so we were able to parse the row information individually. We used python built-in minidom package to parse it.

Also we noticed that jobs to process simple row count in python spark was taking up too much time. We later found out that whole data is contained in single partition. This was because we were using compressed file as our input data source. As compressed file data size is small, and also is a single file, whole uncompression part and then only single executor was making the whole process very slow. So we ran a separate job to uncompress the data and then repartition it to 1000 partitions. We then stored this to separate directory and used that as our data source. We observed a decrease in processing time from 1 hour to 15 minutes.

2.4 Architecture:

Figure 2: High Level Architecture Diagram



2.5 Score Calculation:

We need to calculate score of users of based upon different combination of tags. One way is that every query from user we run spark job to filter out tag specific post and votes, and use them to calculate the score. But spark job needs about 10-15 minutes to calculate the score based on current dataset. So it was decided to preprocess the score and store in the HBase database.

As of now there are about 30000 tags in the StackOverflow, so every possible combination would be close to 230000, which would be impossible to compute to store. So we decided to store only score of individual tags and combine them on the fly based on the user request.

Steps to calculate the score is described below briefly:

1. First we will use the Votes.xml to calculate Total Up votes, Down Votes and Accepted response each post received. We used PostId as key and ran a reduce job to calculate the counters.
2. Then we processed Posts.xml, we first filter out all posts who are not Question or Answer. Then tag information is only present in QuestionPosts, so first filter Posts into 2 subparts QuestionPosts and AnswerPosts. We then join AnswerPosts to QuestionPosts, so that we can map tag information to AnswerPosts as well. Then we take a union to get final Posts.
3. We join the above calculatedPosts with VotesCounter Rdd. Then we calculate the score of each posts, using the counters and the type of post.
4. We then execute a flatmap to create (TagName,UserId) as key and score as value. Then we reduce it calculate the aggregate score.

After we get this score, based upon user query we will join score of different tags to get aggregated score. In order to this to work, scores should be the same scale. This is because on StackOverflow, some tags are very popular so naturally will lead to large scores. So in order to add scores of different tags, we need to convert them to same scale. Here, we are using percentile, to normalize the scale.

After calculation of scores, we are converting this data into 2 rdds. One where key is tag name, which contain top 100 UserIds, and other User rdd where each user and their respective tags score as an array is stored. We now write this information to hbase table.

We weren't able to HBase connector to work directly in spark, so we wrote the results in json format to a text file, and created java yarn job to write data to hbase.

3. Problems Encountered:

- Xml Parsing: Parsing big XML file in spark is difficult. We tried the library, but were not successful. Instead we used minidom in python. We ignored the header in xml and parsed the individual lines containing the row tags.

- While running the jobs, we noticed that jobs were running using single executor because number of partition were one, because of compressed file size format. We tackled this by partitioning the data into 1000 partitions and storing it at separate folder.

- We search someway to connect to HBase and pyspark directly and seamlessly, but weren't able to do so. So we wrote pyspark output to a file and ran yarn java job to write data to database.

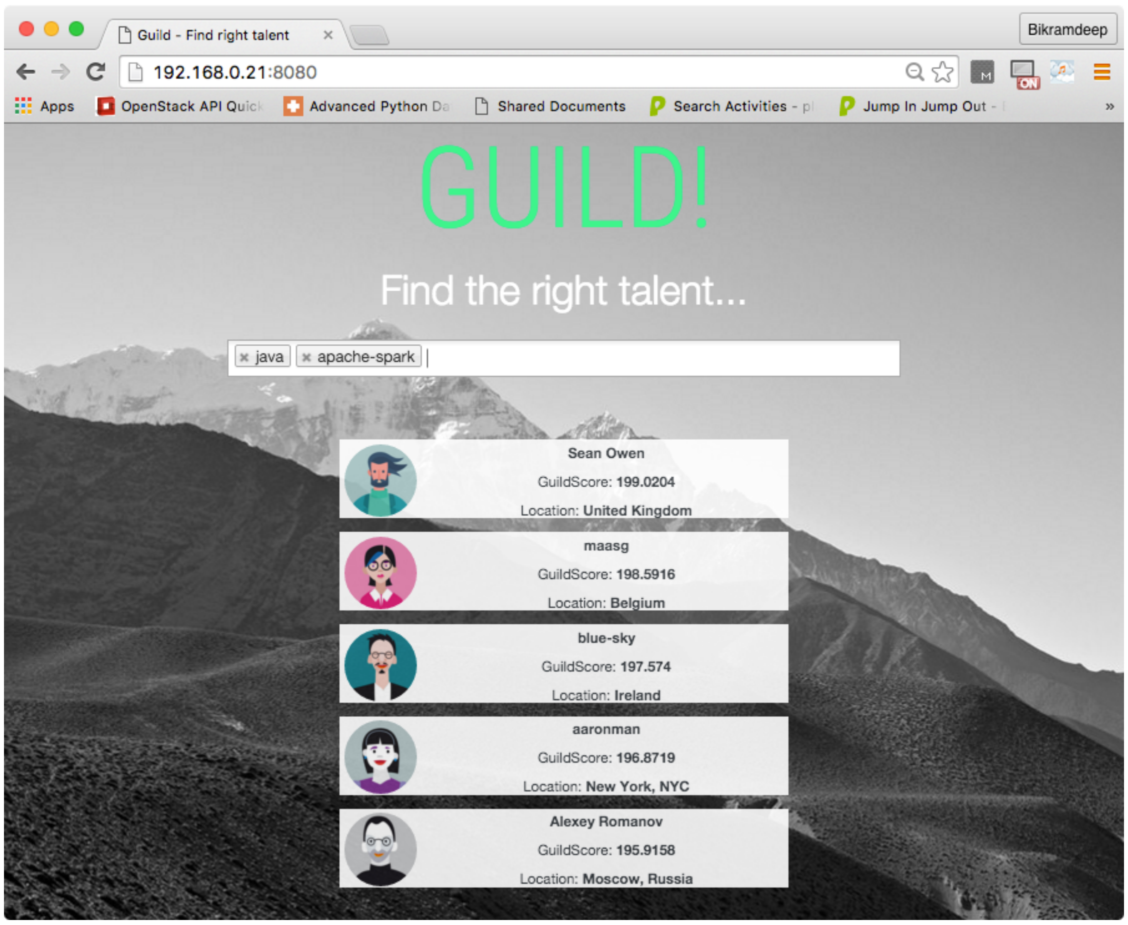
- We first decided to run reduce jobs on the fly based upon user input, so that only user combination of tags are processed. But instead we preprocessed and save the results in HBase so in order to reduce response time. We do multiple tag score calculation on the fly by using HBase, which provides us with fast data access.

- We were not able to not find a way so that the client application could communicate with HBase, we tried some methods but failed, finally we used REST APIs along with Apache Thrift server to expose HBase data via REST.

4. Results:

We were able to create an interactive UI where user can input different skills and our systems returns the top candidates with their respective scores. Over the course of this project we learned various things such as while analyzing very large compressed file, major time is spent on decompression and repartitioning. So it's better to decompress the data into different partitions (assuming there are no space limitations) to speed up the processing work. HBase access is very fast and very easy interfaces are available outside the spark apache framework. HBase can act as a bridging tool between BIG-DATA spark world to traditional technological frameworks.

Figure 3: Screenshot of running app



Sean Owen
GuildScore: 199.0204
Location: United Kingdom

maasg
GuildScore: 198.5916
Location: Belgium

blue-sky
GuildScore: 197.574
Location: Ireland

aaronman
GuildScore: 196.8719
Location: New York, NYC

Alexey Romanov
GuildScore: 195.9158
Location: Moscow, Russia

4.1 Future Enhancements:

- In web UI, we could location filter as well, so that hiring managers can look for candidates in one particular location
- We can use frequent pattern mining to find tags to usually go together, this would aid the users to find skills that used in conjugation
- Right now we give equal weightage to tags given by user, but we could allow the managers to tune out the percentage weight of different tags so that more efficient search is possible.

5. Project Summary

- **Getting the data:** Acquiring/gathering/downloading - **2 points**
- **ETL:** Extract-Transform-Load work and cleaning the data set - **3 points**
- **Problem:** Work on defining problem itself and motivation for the analysis - **2 points**
- **Algorithmic work:** Work on the algorithms needed to work with the data, including integrating data mining and machine learning techniques - **4 points**
- **Bigness/parallelization:** Efficiency of the analysis on a cluster, and scalability to larger data sets - **3 points**
- **UI:** User interface to the results, possibly including web or data exploration frontends - **4 points**
- **Visualization:** Visualization of analysis results - **0 points**
- **Technologies:** New technologies learned as part of doing the project - **2 points**