A Twitter based job search engine using Big Data technologies

Sheenam Gupta
sgupt068@ucr.edu

Krishna Kabi
kkabi004@ucr.edu

Ponmanikandan Velmurugan
pvelm001@ucr.edu

Arun Venkatesh
avenk013@ucr.edu

ABSTRACT

Existence of social network platforms have led to generation of humongous volumes of data. In that aspect, a survey [13] suggests that around 71% of recruiters use social media networks to find and hire job seekers. Due to its high velocity and volume, this information is overwhelming for every job seeker, and many find it very difficult to filter and find job listings that are tailored to their needs. As a result, it creates a potential for a scalable and fault tolerant solution that addresses aforementioned necessities. In an attempt to tap it, we propose a job search engine that leverages state-of-the-art Big Data technologies. The proposed model aids fellow job seekers to find and land their dream job. To make it more user friendly, the model also displays a set of trending jobs that will help users to understand the current demand in the market.

KEYWORDS

Big Data, Search Engine, Twitter Streaming, Spark NLP, Optical Character Recognition(OCR), Distributed Indexing, BM25, TF-IDF, REST

1 INTRODUCTION

The Boom of social media networks in the early 2000s created an ever diverging gap between volume of data generated and available processing power. Social media started as entertainment, and spread its roots into business, e-commerce, and is now omnipresent. It is also a popular platform for job recruiters to identify and connect with potential candidates. However, most social media platforms lack native support for job seekers and recruiters. Therefore, we address this issue by creating a scalable and fault tolerant job search platform that runs on top of the Twitter network and provides relevant job listings based on a user query.

Since a job search engine needs a lot of data to work with, we have used Twitter as our source to filter job listings. In order to achieve that, we have used Tweepy, a wrapper over Twitter Developer APIs, to extract live streaming Tweets and store it locally on a regular basis. These data are then refined in the pre-processing stage by filtering problem specific parameters through Apache Spark, modeled using Solr, ranked using BM25 and TF-IDF algorithms and finally exposed as APIs to the web application. The data obtained from Twitter's live streaming contains a lot of noise, such as sensitive content, that needs to be eliminated. Such noise is eliminated in a custom data preprocessing stage that makes use of Spark NLP pipeline. Along with this, we have also implemented a OCR model that loads the tweet media content and provides text translations. These texts will be used in later stages to further improve the ranking model.

In the data modelling phase, we create a pipeline to load, transform and index the batched data using Apache Solr. The scalable search engine offers advantages such as distributed indexing, re-indexing, schema based optimizations and field based queries which maintains the correctness and improves the usability and performance of the module. At last, the back end application fetches the filtered data from the ranking model and exposes them as REST APIs to the web application. The back end is written in Java and runs on a Spring Boot server. On the other hand, the web application, written in AngularJS, provides a interactable, user-friendly interface for potential job seekers to search for a job based on multiple parameters such as query, user and location.

The rest of the paper contains the above discussed components in detail. Section 2 reviews the existing solutions, followed by the Analytical framework that explains the proposed solution's working based on its different modules in section 3. Further, we have validated the performance and the scalability of the model by providing comparative analysis to get a better insight on the working model in section 4. The current model is visualized, and described in section 5 in order to provide better understanding for the readers. Finally in section 6, we have summarized all the components briefly in the conclusion along with the possible future work in this area.

2 RELATED WORK

In the following subsections, we will be addressing the existing work in each of the project's categories and discuss in brief how we learn and use them to better solve the problem at hand.

2.1 Data Gathering

Article[1] proposes a basic sentiment analysis of data obtained from Twitter's historic search API and gives a simple pie chart based on the sentiment percent. Article [2] introduces a machine learning algorithm that uses Twitter's historic search API by filtering based on keywords such as 'job', 'hiring', 'vacancy' and 'employment'. Article [3] describes the HF-IHU (Hashtag frequency-Inverse Hashtag Ubiquity) ranking algorithm that recommends content to a user based on their interests. This uses Twitter API as well but specifically works only on hashtags. The above related work helps greatly in identifying the unique keywords that can be used to retrieve recruiting information. It also allows us to handle the drawbacks of using Twitter's historic API and to not restrict the ranking model based on hashtags.

2.2 Data Preprocessing

Article [4] talks about the effectiveness of using OCR in extracting text from images compared to the Transym package. We use a different version of tesseract as a tool to extract text from images and attach it to the main tweet. Similarly, in the data preprocessing phase, articles [5] and [6] present various methods used for populating trending topics such as LDA, K-means clustering, Jaccard Similarity etc. Our initial approach was to find top trending words by taking a ratio of current word frequencies over the history in a certain time period (usually within 15 minutes window) and also use chi-square test to further filter trending words that pass a certain threshold. But we focussed on using SparkNLP for word cloud generation and populating the top 4-5 words from each batch.

2.3 Data Modeling

Articles [7] and [9] describe the various inherent features of Apache Solr that can be used to improve the search engine relevance of the application. Article [7] specifically describes the use of token analyzers to process the query term better. These techniques are collectively used in our project to improve the accuracy/relevance of the query results. Article [9] describes the advanced features offered by Apache Solr such as query highlighting, query auto-complete, facet navigation. These features can improve the user experience of the application. Paper [8] proposes various methods to perform topic detection and tracking with twitter data while addressing key challenges and evaluation issues.

2.4 Data Visualization

Article [10] works on retrieving 2 lists; One is focussed for the job seeker with matching keywords from their Resume and the other to the organization showing the suitable candidates for the job role. Article [11] works on taking the resume as input and displaying results based on keyword match and similarity scores. Article [12] proposes a job recommendation system by representing the end results in the form of a job-job relationship graph, where nodes are the job with multiple association edges.

3 ANALYTICAL FRAMEWORK

In the following subsections we will discuss in detail the various aspects of the proposed solution. The proposed solution consists of five different components which are depicted in figure 1.

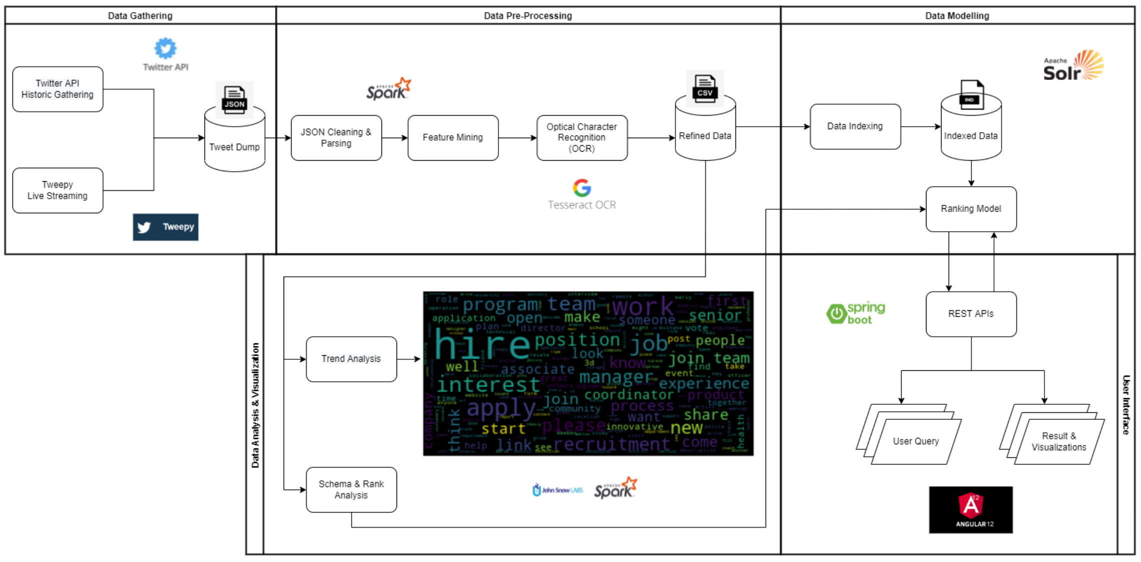


Figure 1: Project Overview

3.1 Data Gathering

This section describes the first phase of the proposed model, the data collection framework. This component consists of collecting live streaming Tweets on Twitter and storing it locally. The frameworks that are used in this module as follows:

- Python - Base programming language
- Tweepy - Wrapper library over Twitter's API framework

This framework supports configurable parameters such as the following:

- api_key = API Key of the Twitter Developer account's app
- api_secret = API secret of the Twitter Developer account's app
- access_token = Access token of the Twitter Developer account's app
- access_token_secret = Access token secret of the Twitter Developer account's app
- time_limit = Overall run time duration of the framework (Denotes the total number of seconds for which the data has to collected) (in seconds)
- num_retries = Number of times the code should restart on exceptions or errors (Failure handling)

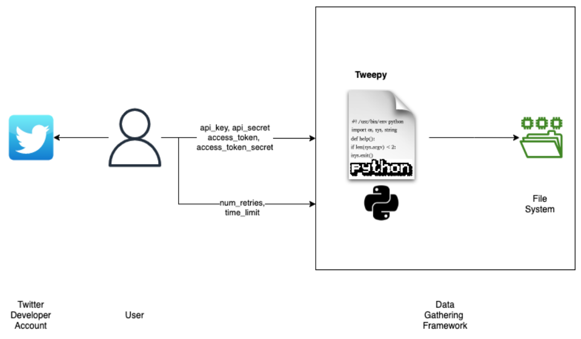


Figure 2: Overall Flow - Data Collection

The basic workflow as explained in Figure 2 is as follows:

- The api_key, api_secret, access_token, access_token_secret is obtained from Twitter developer account portal for an app
- The time_limit and num_retries is set accordingly (in seconds)
- These are configured in the Python library in their necessary placeholders
  The Python script is invoked which runs till the given time_limit from the execution time
- The file name is set in the following format : output<retry-count>.json

Figure 3 explains the functionalities of the configurable parameters that are supported: num_retries and time_limit. The live streaming Tweets are obtained by making use of Tweepy's stream API, which is a wrapper over Twitter's Live Streaming API. The streamed data is searched for the keywords 'hiring' and 'recruitment' and the results are stored as a JSON at the end of the program's execution.

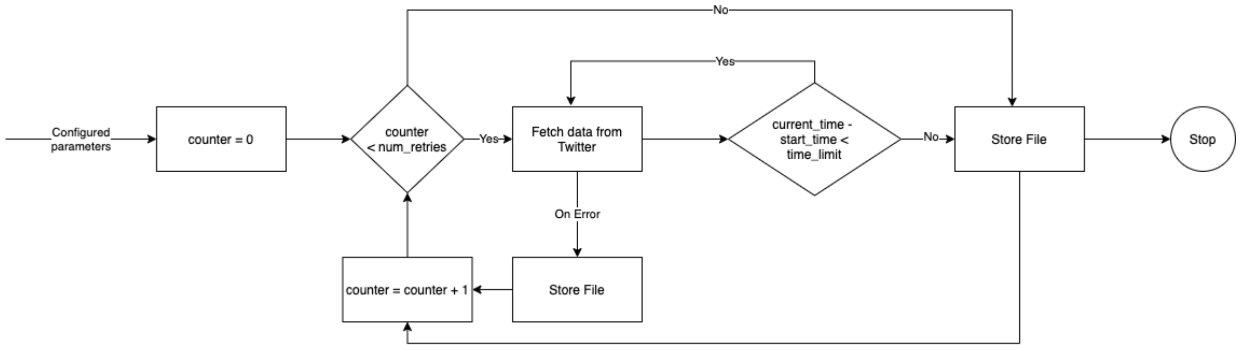


Figure 3: FlowChart - Data Collection

3.2 Data Description

Any streaming tweet data has the original tweet, the user details who posted it, user mentions along with images and links and hashtags. The twet object also mentions if it's a retweet or quoted tweet which is of particular interest in our case. The text of the field is of vital importance as this is used to build an index on Solr. But to provide overall user experience, we also populate that particular tweet along with user related information such as name, image, followers count, likes count, user mentions etc.

3.3 Data Preprocessing

In the following sections we will briefly discuss the strategies involved in data pre-processing.

3.3.1 Data Munging

Since we deal with large volumes of unstructured data a simple json parser although would work but would consume significant execution time. We therefore opted to use a big data system such as Spark. What makes parsing of tweet data complex is the data dictionary that changes with the type of tweet. For instance, the base level or original tweet had different fields to extract compared to a tweet which is a retweet or a quoted tweet. A glimpse of such processing is summarized in figure 4.

The inbuilt functionalities of spark such as inferring schema of json, explode, UDFs made traversing the data in nested structure easier. Since we were running spark on local, processing of larger files was a bottleneck initially as it threw “out of memory error”. This was the main bottleneck for our processing. So, during our initial phase, we were able to process around 50MB files because of that. On analysis we could find several ways to tweak memory to enhance our processing capability. For instance, the initial spark configuration has a shuffle partition as 200. This creates many partitions and few threads to process with very less data. Changing the partition size to a reasonable number based on file size resolved certain out of memory issues yet not entirely. We had to increase driver memory allocation and make use persist and unpersist to utilize the memory assigned appropriately. So, for cases where we relied on previous data, we persisted it in memory and disk and unpersisted when we no longer required it. Combining all such tuning helped us in processing a bigger chunk of file.

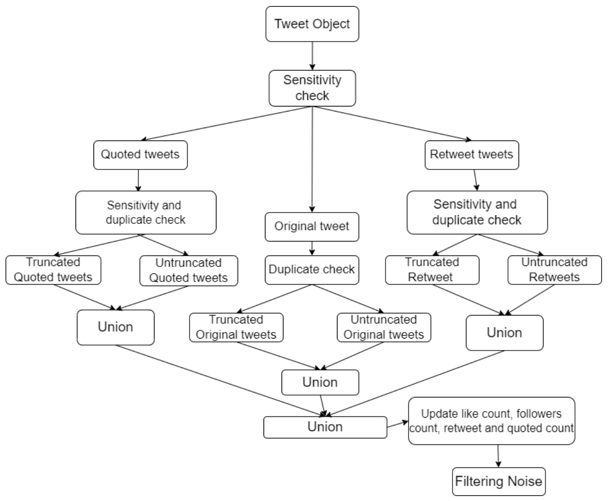

Tweet Object
Sensitivity check
Quoted tweets
Original tweet
Retweet tweets
Sensitivity and duplicate check
Sensitivity and duplicate check
Truncated Quoted tweets
Untruncated Quoted tweets
Duplicate check
Truncated Retweet
Untruncated Retweets
Union
Truncated Original tweets
Untruncated Original tweets
Union
Union
Union
Update like count, followers count, retweet and quoted count
Filtering Noise

Figure 4: Data Preprocessing

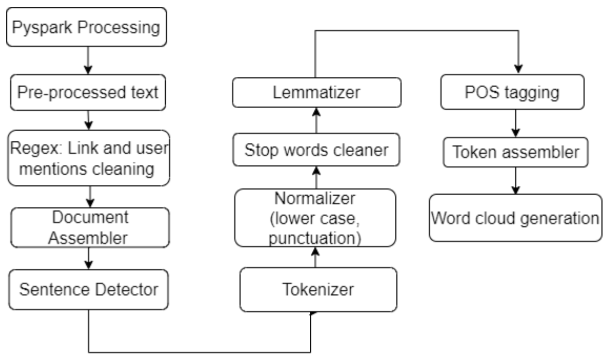


Figure 5: NLP Pipeline

3.3.2 Data Cleaning

Data collected with keywords hiring and recruitment although helped getting data related to hiring, it also picked up a lot of noise. Few Tweets particularly had sensitive content which the twitter object tag with sensitive flag. This had to be checked at original or outer level, retweet level and quoted level.

After parsing data, the data is in tabular format suitable for downstream objects. But our initial analysis showed there were many tweets which although had the text hiring/recruitment but had very different context to it unrelated to our purpose. So filtering out those tweets required analysis on a sample of data and selecting phrases appropriate for hiring such as “are hiring now, recruiting now, apply now” etc and 30 equivalent forms of it. Spark regex functionality was used to do such filtering.

3.4 Data Analysis

In the following sections we will briefly discuss the strategies involved in data analysis.

3.4.1 Trend Analysis

The trending hashtag or trending keyword had more to do with how we pre-process the text of the tweet. This field would help the user to know what particular hashtag or keywords related to certain words are trending. To extract words from text we used SparkNLP library to preprocess the text. The NLP pipeline adapted for our case is shown in Figure 5.

The processing is standard to any text processing. We first detect documents, then split them into sentences followed by tokenizing words, stop words removal, lemmatizing. To get trending keywords, we are only concerned about nouns. SparkNLP provides POS(part of speech tagging) functionality to detect nouns, adjective verbs etc. We filter the text based on these tagging and use word clouds to populate most commonly used words. We process this for every batch of records and get top frequent words trending in that batch. However, we could find the keywords extracted were not that informative for a front end user point of view but we still included it for the sense of completeness.

3.4.2 Solr Schema & Rank Analysis

In this section we will cover the schema possibilities and their impact on the overall retrieval performance of the Apache Solr search platform. Apache Solr offers customizations in the following to tailor the retrieval model to our problem statement:

- Solr FieldType
- Solr Field

We will be discussing each of them in detail in the following subsections

3.4.2.1 Solr FieldType

The FieldType is the one of the simplest units of Solr. It describes how a dataset (or) query feature is defined, tokenized and ranked. Table 1 shows the base parameters needed to create a Solr FieldType. The class [14] defines the feature and offers additional type specific operations for some of them. For example, using DatePointField will allow us to query the field with a date range. In practice, This retrieves all the tweets that fall within the timestamp range. Such proper feature mapping improves the usability and efficiency of the retrieval process. The tokenizers [15] are used to determine how the fields are tokenized during the indexing phase. For example, KeywordTokenizer can tokenize the entire field value as a single token which can be useful for fields like username where the whitespace based tokenization will be meaningless. The similarity class [16] determines how the tokenized field will be ranked. For example, ClassicSimilarity is a ranking algorithm based on the Vector Space Model's Cosine Similarity. However, due to the nature of the project, we will be using the BM25 ranking algorithm for all the fields because of its efficiency in ad-hoc retrieval i.e, the BM25 ranking shines when the search span is wide like in job search, where users have a broad spectrum of search queries. Therefore, proper mapping of tokenizer with the ranking algorithm guarantees retrieval correctness.

3.4.2.2 Solr Field

The Field [17] is the second most simplest unit of Solr. It extends the FieldType and defines how the field will be, treated during, and stored after the indexing process . Table 2 shows the base parameters needed to create a Solr Field. The Solr FieldType is discussed in the earlier section. If a FieldType is not predefined, we can create our own types and assign them. The indexed and stored are boolean fields which determine how the field should be processed. We can choose to just index the field and not store them, this is coupled with database lookup to retrieve the raw data. However, considering the narrowed scope of the project we both index and store the field values. The muliValued and docValues are additional parameters that determine how the fields are stored. For example, multiValued stores fields having more than one value and offers retrieval based on single or multiple field value. This improves the usability of the retrieval.

Parameter	Comment
name	FieldType name
class	Solr field class
indexAnalyser	Provide field tokenizer
queryAnalyser	Provide query tokenizer
similarity	Solr similarity class

Table 1: Solr FieldType Schema Parameters

Parameter	Comment
name	Field name
type	Field type
indexed	Will index the field if true
stored	Will store the field if true
multiValued	Field have multiple values if true
docValues	Efficient for field that require sorting and faceting

Table 2: Solr Field Schema Parameters

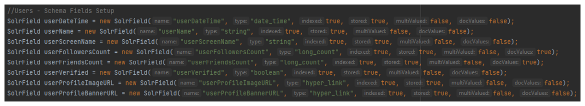
In our project, the hashtag search API, which will be discussed in section 3.5.3.3, is possible because of this concept. The docValues are used to store fields that require sorting and faceting. This improves the index storage and lookup efficiency by exploiting the column based strategies. Thus understanding the simplest units of Apache Solr Search platform helps us in improving the usability, storage, and retrieval efficiency for the problem at hand. This detailed analysis will be used during each part of the Data Modelling phase to optimize the model.

3.5 Solr Data Modeling

Apache Solr is a real-time, open source enterprise search platform built on Apache Lucene. The key advantages of Apache Solr is scalable distributed indexing which offers index replication and fault-tolerance. In the following subsections we will be defining the solr schema, indexing the dataset, and performing efficient retrieval by leveraging the aforementioned problem specific schema and indexing standards.

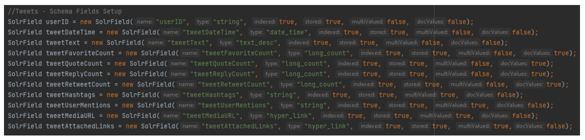
3.5.1 Solr Schema Definition

In this section, we will review the schema decisions made for the project based on the analysis performed on section 3.4.2. Each dataset record is made up of Tweet and User objects. The Solr Fields created for user objects are depicted in figure 6. Based on the analysis, the numeric fields are mapped to LongPointField and stored as docValues to improve the search options, storage and retrieval efficiency. The date fields are mapped to DatePointField which offers raw timestamp indexing and range retrieval functionalities. The user_verifed field is stored in native boolean datatype, user_profile_image and user_banner_image are stored as TextField which offers efficient URL tokenization, for optimized retrieval.



```java
//Users - Schema Fields Setup
SolrField userDateTime = new SolrField( name: "userDateTime", type: "date_time", indexed: true, stored: true, multiValued: false, docValues: false);
SolrField userName = new SolrField( name: "userName", type: "string", indexed: true, stored: true, multiValued: false, docValues: false);
SolrField userScreenName = new SolrField( name: "userScreenName", type: "string", indexed: true, stored: true, multiValued: false, docValues: false);
SolrField userFollowersCount = new SolrField( name: "userFollowersCount", type: "long_count", indexed: true, stored: true, multiValued: false, docValues: true);
SolrField userFriendsCount = new SolrField( name: "userFriendsCount", type: "long_count", indexed: true, stored: true, multiValued: false, docValues: true);
SolrField userVerified = new SolrField( name: "userVerified", type: "boolean", indexed: true, stored: true, multiValued: false, docValues: false);
SolrField userProfileImageURL = new SolrField( name: "userProfileImageURL", type: "hyper_link", indexed: true, stored: true, multiValued: false, docValues: false);
SolrField userProfileBannerURL = new SolrField( name: "userProfileBannerURL", type: "hyper_link", indexed: true, stored: true, multiValued: false, docValues: false);
```

Figure 6: User Schema Definition



```java
//Tweets - Schema Fields Setup
SolrField userID = new SolrField( name: "userID", type: "string", indexed: true, stored: true, multiValued: false, docValues: false);
SolrField tweetDateTime = new SolrField( name: "tweetDateTime", type: "date_time", indexed: true, stored: true, multiValued: false, docValues: false);
SolrField tweetText = new SolrField( name: "tweetText", type: "text_desc", indexed: true, stored: true, multiValued: false, docValues: false);
SolrField tweetFavoriteCount = new SolrField( name: "tweetFavoriteCount", type: "long_count", indexed: true, stored: true, multiValued: false, docValues: true);
SolrField tweetQuoteCount = new SolrField( name: "tweetQuoteCount", type: "long_count", indexed: true, stored: true, multiValued: false, docValues: true);
SolrField tweetReplyCount = new SolrField( name: "tweetReplyCount", type: "long_count", indexed: true, stored: true, multiValued: false, docValues: true);
SolrField tweetRetweetCount = new SolrField( name: "tweetRetweetCount", type: "long_count", indexed: true, stored: true, multiValued: false, docValues: true);
SolrField tweetHashtags = new SolrField( name: "tweetHashtags", type: "string", indexed: true, stored: true, multiValued: true, docValues: false);
SolrField tweetUserMentions = new SolrField( name: "tweetUserMentions", type: "string", indexed: true, stored: true, multiValued: true, docValues: false);
SolrField tweetMediaURL = new SolrField( name: "tweetMediaURL", type: "hyper_link", indexed: true, stored: true, multiValued: true, docValues: false);
SolrField tweetAttachedLinks = new SolrField( name: "tweetAttachedLinks", type: "hyper_link", indexed: true, stored: true, multiValued: true, docValues: false);
```

Figure 7: Tweet Schema Definition

The Solr Fields created for tweet objects are described in figure 7. The tweet hashtags, user_mentions, and tweet_media are stored as multiValued for value based retrieval and StrField for single value tokenization and meaningful retrieval. To summarize, by leveraging proper schema mapping (field and field types) mapping we sustain the correctness, improve querying opportunities and retrieval performance.

3.5.2 Solr Data Indexing

In this section, we will be discussing the solr indexing strategies in brief. First, we need to understand the nature of input for Solr Platform. After the data cleaning process, discussed in section 3.3.2, we obtain CSVs in batches as input for Solr. Each record of these CSV contains a Tweet object and a corresponding User object. The first level of optimization is done here where we will be using hashmaps to identify and store only the unique and latest user objects in the batch. This is done with the assumption that a single user can create multiple tweets and for each tweet the user object is repeated. Raw storage of such user objects would just increase the index size and would be meaningless because the record does not track the latest user object after indexing. The second level of optimization is achieved with Solr's built-in reindexing mechanism, where User and Tweet objects that already exist in the respective collections will be replaced with the new objects from the batch based on unique identifiers. This preserves the correctness of the solr module, and does not waste index size as new batches of data are received. The schema for these collections are discussed in brief in previous section 3.5.1. Figure 8 depicts the base aspects of the indexing process.

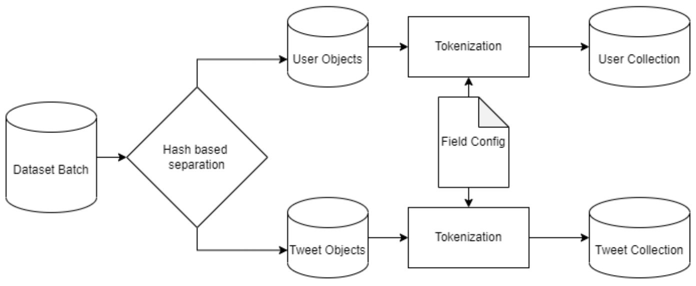


Figure 8: Indexing Flowchart

3.5.3 Solr Ranking Model

In this section, we will be providing pre-defined retrieval commands which can be used by application users. As discussed earlier, due to the narrowed scope of the project, we are indexing and storing the data using Apache Solr. The retrieval commands provided include:

- Keyword Search (Base)
- User Search
- Hashtag Search

We will be discussing each command in detail in the following subsections. These commands are wrapped in Spring Boot which will be discussed in section 3.6.1.

3.5.3.1 Keyword Search

The keyword search is the basic search where the tweet collection's tweet_text field is queried and the results retrieved in descending order of BM25 rank score. Then, we will obtain the user_id from the results to boolean query the user collection and gather corresponding user objects. These two objects are joined to obtain the original record. Due to the reindexing functionality both the tweet and user object would contain the latest metrics such as favorite_count, and followers_counts. This command powers the main functionality of the project where the user will query job titles and get relevant job postings.

3.5.3.2 User Search

This is a specific command to obtain just the user object and the user's list of tweet objects. This is used to recreate the user profile, in our case the recruiters profile and their list of recruitment tweets. First, we will query the user collection, gather the user object and then boolean query the tweet collection's user_id field and order the obtained tweet objects based on timestamp. The timestamp based ordering is possible because of the use of DatePointField.

3.5.3.3 Hashtag Search

This is a specific command used to retrieve all the tweet objects based on a hashtag. Similar to base keyword search, first we obtain all the tweet objects and gather the corresponding user objects. However, we query the tweet collection based on the multiValued tweet_hashtag field and not the tweet_text field. This result is again ordered based on timestamp to have the latest tweet first. This is similar to hashtag search offered by twitter. This search is possible because of the multiValued nature of the field and Solr offering value based search on them.

Thus we have covered the dataset features, corresponding Solr mapping (maintains correctness, reduces indexing size, and offers field specific retrieval opportunities), efficient indexing (based on separation and reindexing) and highly optimized retrieval techniques which leverages the mentioned standards to boost retrieval performance.

3.6 Web Application

In this part, we will be outlining the end goal for our project and explaining how we gathered all the project data to visualize it at one place using different technologies and frameworks.

3.6.1 Backend

The backend for the proposed solution is a layer that provides a set of REST APIs for the web application (Section 3.6) to interact. It communicates with the Solr Ranking model (Section

3.5.3), converts it into a neatly formatted JSON and sends it as a response to the web application. The basic dataflow of the proposed client-server model is shown in Figure 9. The backend is a Maven based project, written completely in Java, running on Spring Boot. The APIs are exposed using the @RestController annotation supported by the Spring Boot framework. The list of APIs that were built to support the web application is explained in Table 3 and a detailed view of the API's response parameters and their purpose are explained in Table 4.

A sample scenario would be as follows:

- A user searches for the job titled "Developer"
- The query "Developer" is sent to the backend as a query parameter to the Search API
- The server receives the request, parses the query and validates it
- The query is sent as a parameter to the ranking model that fetches the data from the indexed data from Solr and returns back to the server
- The server converts it based on the output format and returns it back to the web application.

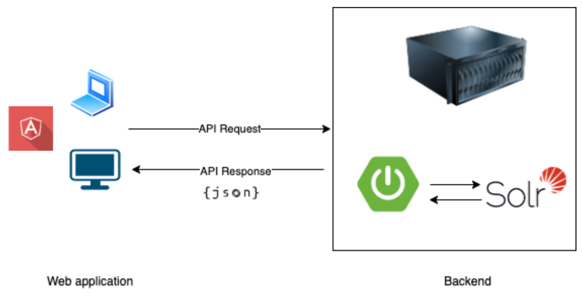


Figure 9 : DataFlow - Web Application and Backend

Example Request :

curl http://localhost:8080/api/search?query=Developer'

Example Response (Structure):

```
[
  {
    "tweet":{
        <tweet-parameters>
    },
    "user":
        <user-parameters>
  }
]
```

Name	URI	HTTP Method	Purpose	Request	Response
Search API	/api/search	GET	Find relevant jobs based on a given keyword	Keyword (Job Title/ Description / Location)	JSON objects that comprises of Tweets and its user data (owner of that Tweet)
User API	/api/user/{user-id}	GET	Get Tweets posted by a certain user	Twitter User ID	JSON Array of Tweets posted by that user

Hashtag API	/api/hashtag	GET	Get Tweets and its user meta based on a hashtag	Hashtag keyword	JSON objects that comprises of Tweets and its user data (owner of that Tweet)

Table 3: API Documentation - Back End

Name	Purpose
tweet.tweetQuoteCount	Number of times the Tweet was quoted
tweet.tweetRetweetCount	Number of times the Tweet was retweeted
tweet.tweetReplyCount	Number of replies for the Tweet
tweet.tweetDateTime	The time and date in which the Tweet was posted (datetime format)
tweet.tweetAttachedLinks	Attachments that were a part of the Tweet (Array)
tweet.tweetText	Tweet's message
tweet.tweetFavoriteCount	Number of times the Tweet was marked as favorite
tweet.userID	ID of the user who posted the Tweet
tweet.tweetID	Unique identifier to identify the Tweet
tweet.tweetUserMentions	The list of users who were mentioned in the Tweet (Array)
tweet.tweetMediaURL	The list of media attachments that were a part of the Tweet (Array)
tweet.tweetHashtags	The list of hashtags used to tag that Tweet (Array)
user.userFollowersCount	Number of followers for the user
user.userProfileImageURL	Hosted URL of the user's profile image
user.userDateTime	The date and time in which the user signed up the account
user.userFriendsCount	Number of friends that the user has
user.userScreenName	Screen Name of the user
user.userProfileBannerURL	Hosted URL of the user's banner image
user.userVerified	Boolean to determine if the user's account is verified

user.userName	User Name of the user
user.userID	Unique identifier to identify the user

Table 4: API Response Parameters

3.6.2 Front-End & Visualization

We are using AngularJS, version 12, for our frontend visualization. It is hosted locally, backend and frontend are currently hosted locally running on different ports.

The following are the features that are offered by the front-end:

- Search Engine Page - This includes the search box, query based results showing Tweets and the information on users who posted that Tweet. On the right side, there will be a display of the current trending jobs. By default the home page will be empty and gets updated asynchronously when a user searches for a query. In the search based query model, results are displayed according to the matching keywords for the given query.
- Trending Keywords page - A list of trending job titles with some description on each job title. Currently, this is a static list that is generated from the word cloud image.
- User Profile page - This contains information about user meta and statistics such as total followers, total Tweet count, banner image, along with user meta like username, and their basic profile details. It also displays all the Tweets of a particular user in a single place, also including their like count, images and hashtags.
- Hashtags Page - Since almost every tweet has a hashtag associated with it, we have built a hashtag page separately to see all related tweets based on a particular hashtag. This will ensure a group of Tweets are bundled together to provide a more cohesive output to the user.
- Additional validations such as “No data found” checks are handled gracefully in the application to notify users that there is no data for a particular search mode.
- Apart from these, since the volume of data might be high, and to be scalable, the model also supports pagination to ensure that data and load is divided equally on all pages that also increases user engagement experience.

4 EVALUATION

In the following sections, we will discuss the evaluation methods used in the project. The evaluations are done on Spark to validate and measure the parallelizability and performance.

4.1 Performance vs Input Size

We processed 100MB, 200MB, 400MB, and 800MB files with 8 partitions and 8 cores assigned. The execution time is as shown in Figure 10. With increase in file size we do see increase in execution time. The plot shows that this can be scalable as the curve has a sub-linear growth to it.

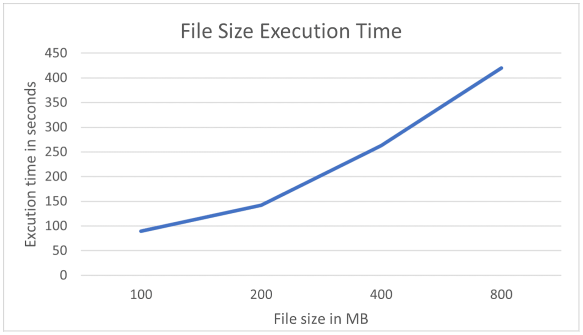


Figure 10: File size vs execution time

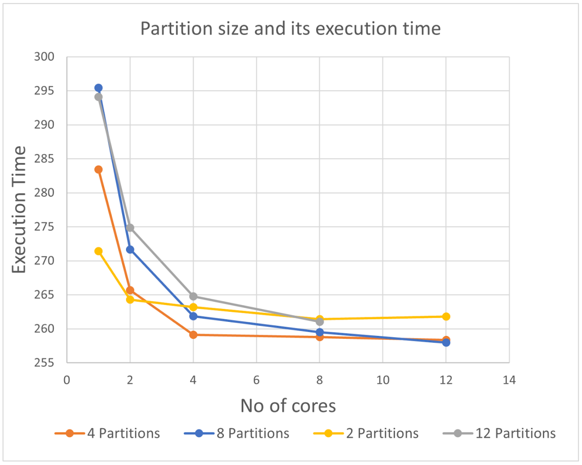


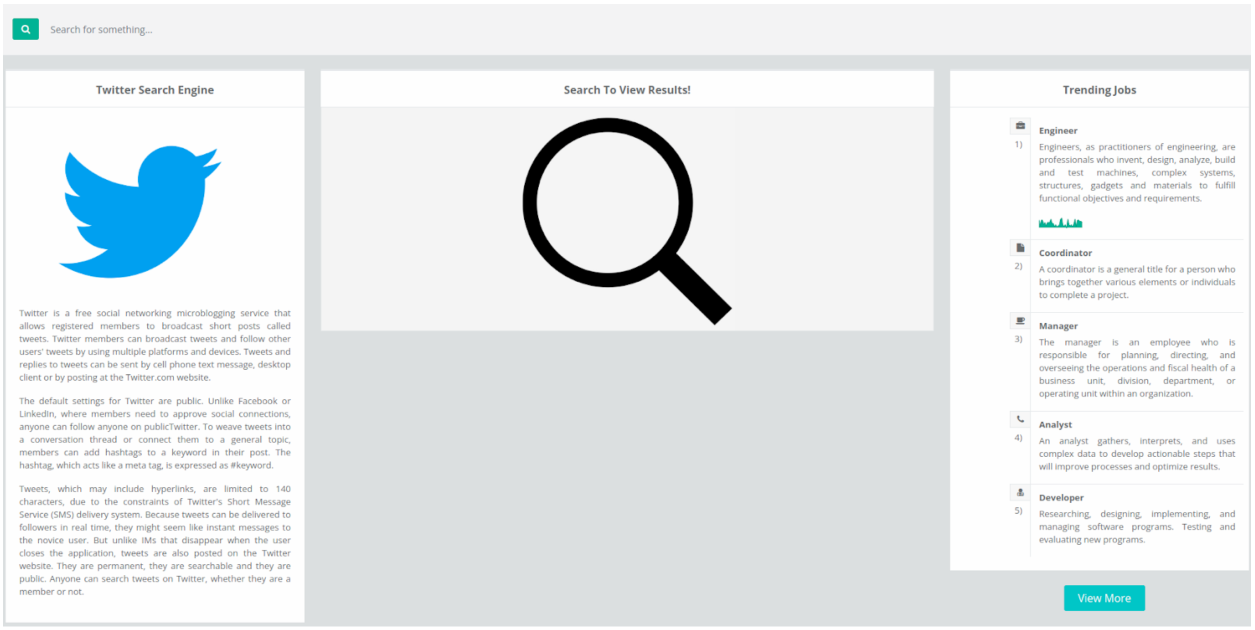
Figure 11: Partitions vs Execution time

4.2 Performance vs Parallelizability

For evaluating how a file sized processing is affected by varying number of partitions and cores, we used a 400MB file and evaluated it with partitions 2, 4, 8 and 12. The execution time taken is as shown in figure 11. With an increase in the number of threads the execution time decreases. Ideally, assigning 1 core per partition helps in achieving true parallelism. Whereas, too few partitions can lead to out of memory issues and too many partitions can have scheduling issues in the executor.

5 RESULTS

In the section, we will be attaching the screenshots of the working model and explain the moving variables associated with each of them in detail.



Twitter is a free social networking microblogging service that allows registered members to broadcast short posts called tweets. Twitter members can broadcast tweets and follow other users' tweets by using multiple platforms and devices. Tweets and replies to tweets can be sent by cell phone text message, desktop client or by posting at the Twitter.com website.

The default settings for Twitter are public. Unlike Facebook or LinkedIn, where members need to approve social connections, anyone can follow anyone on publicTwitter. To weave tweets into a conversation thread or connect them to a general topic, members can add hashtags to a keyword in their post. The hashtag, which acts like a meta tag, is expressed as #keyword.

Tweets, which may include hyperlinks, are limited to 140 characters, due to the constraints of Twitter's Short Message Service (SMS) delivery system. Because tweets can be delivered to followers in real time, they might seem like instant messages to the novice user. But unlike IMs that disappear when the user closes the application, tweets are also posted on the Twitter website. They are permanent, they are searchable and they are public. Anyone can search tweets on Twitter, whether they are a member or not.

1) **Engineer**
Engineers, as practitioners of engineering, are professionals who invent, design, analyze, build and test machines, complex systems, structures, gadgets and materials to fulfill functional objectives and requirements.

2) **Coordinator**
A coordinator is a general title for a person who brings together various elements or individuals to complete a project.

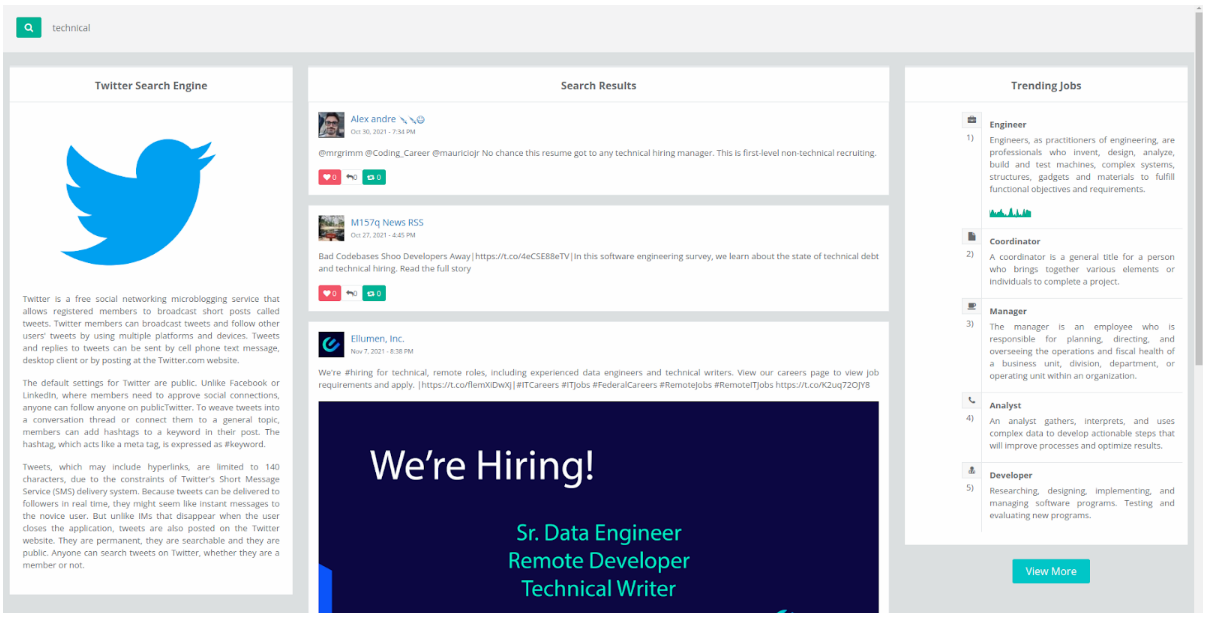
3) **Manager**
The manager is an employee who is responsible for planning, directing, and overseeing the operations and fiscal health of a business unit, division, department, or operating unit within an organization.

4) **Analyst**
An analyst gathers, interprets, and uses complex data to develop actionable steps that will improve processes and optimize results.

5) **Developer**
Researching, designing, implementing, and managing software programs. Testing and evaluating new programs.

Figure 12: Search Engine Home Page

Figure 12 shows the home page of the search engine. The page consists of three main parts that includes search bar, result section, and trending section. With respect to the user query on the search bar, the result section will be updated. The twitter description is a static section that provides a trivia for Twitter. The trending section is also static based on the word cloud generated on the data preprocessing stage. This was discussed in section 3.4.1


technical

Twitter Search Engine

Twitter is a free social networking microblogging service that allows registered members to broadcast short posts called tweets. Twitter members can broadcast tweets and follow other users' tweets by using multiple platforms and devices. Tweets and replies to tweets can be sent by cell phone text message, desktop client or by posting at the Twitter.com website.

The default settings for Twitter are public. Unlike Facebook or LinkedIn, where members need to approve social connections, anyone can follow anyone on publicTwitter. To weave tweets into a conversation thread or connect them to a general topic, members can add hashtags to a keyword in their post. The hashtag, which acts like a meta tag, is expressed as #keyword.

Tweets, which may include hyperlinks, are limited to 140 characters, due to the constraints of Twitter's Short Message Service (SMS) delivery system. Because tweets can be delivered to followers in real time, they might seem like instant messages to the novice user. But unlike IMs that disappear when the user closes the application, tweets are also posted on the Twitter website. They are permanent, they are searchable and they are public. Anyone can search tweets on Twitter, whether they are a member or not.

Search Results

Alex andre
Oct 30, 2021 - 7:34 PM

@mrgrimm @Coding_Career @mauriciojr No chance this resume got to any technical hiring manager. This is first-level non-technical recruiting.

0 0 0

M157q News RSS
Oct 27, 2021 - 4:45 PM

Bad Codebases Shoo Developers Away|https://t.co/4eCSE88eTV|In this software engineering survey, we learn about the state of technical debt and technical hiring. Read the full story

0 0 0

Ellumen, Inc.
Nov 7, 2021 - 8:38 PM

We're #hiring for technical, remote roles, including experienced data engineers and technical writers. View our careers page to view job requirements and apply. |https://t.co/flemXiDwXj|#ITCareers #ITJobs #FederalCareers #RemoteJobs #RemoteITJobs https://t.co/K2uq72OJY8

Trending Jobs

1) **Engineer**
Engineers, as practitioners of engineering, are professionals who invent, design, analyze, build and test machines, complex systems, structures, gadgets and materials to fulfill functional objectives and requirements.

2) **Coordinator**
A coordinator is a general title for a person who brings together various elements or individuals to complete a project.

3) **Manager**
The manager is an employee who is responsible for planning, directing, and overseeing the operations and fiscal health of a business unit, division, department, or operating unit within an organization.

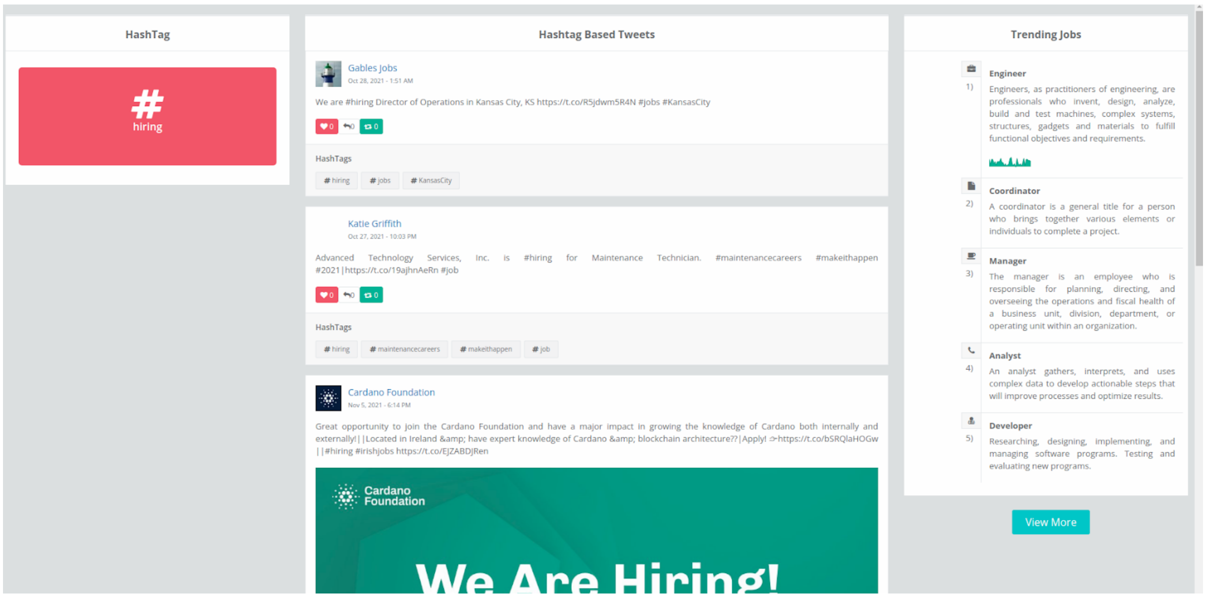
4) **Analyst**
An analyst gathers, interprets, and uses complex data to develop actionable steps that will improve processes and optimize results.

5) **Developer**
Researching, designing, implementing, and managing software programs. Testing and evaluating new programs.

View More

Figure 13: Search - Based on Query - "technical"

Figure 13 below is a snapshot of the engine showing the results of a query based search. As we can see in the image, the query that was searched was "technical" and the results display all the data that correlates to the search query. This is achieved by making use of the Search API as mentioned in Table 3.


HashTag

Hashtag Based Tweets

Gables Jobs
Oct 28, 2021 - 1:51 AM

We are #hiring Director of Operations in Kansas City, KS https://t.co/R5jdwm5R4N #jobs #KansasCity

0 0 0

HashTags
# hiring # jobs # KansasCity

Katie Griffith
Oct 27, 2021 - 10:03 PM

Advanced Technology Services, Inc. is #hiring for Maintenance Technician. #maintenancecareers #makeithappen #2021|https://t.co/19ajhnAeRn #job

0 0 0

HashTags
# hiring # maintenancecareers # makeithappen # job

Cardano Foundation
Nov 5, 2021 - 6:14 PM

Great opportunity to join the Cardano Foundation and have a major impact in growing the knowledge of Cardano both internally and externally!||Located in Ireland &amp; have expert knowledge of Cardano &amp; blockchain architecture??|Apply! ➩https://t.co/bSRQlaHOGw||#hiring #irishjobs https://t.co/EJZABDJRen

Trending Jobs

1) **Engineer**
Engineers, as practitioners of engineering, are professionals who invent, design, analyze, build and test machines, complex systems, structures, gadgets and materials to fulfill functional objectives and requirements.

2) **Coordinator**
A coordinator is a general title for a person who brings together various elements or individuals to complete a project.

3) **Manager**
The manager is an employee who is responsible for planning, directing, and overseeing the operations and fiscal health of a business unit, division, department, or operating unit within an organization.

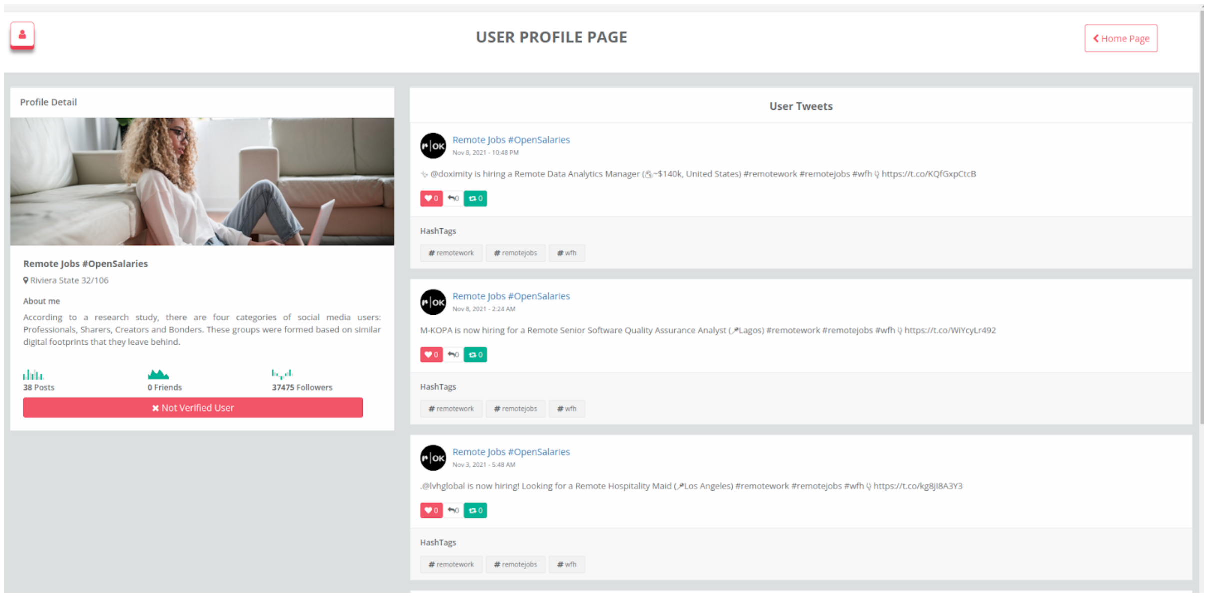
4) **Analyst**
An analyst gathers, interprets, and uses complex data to develop actionable steps that will improve processes and optimize results.

5) **Developer**
Researching, designing, implementing, and managing software programs. Testing and evaluating new programs.

View More

Figure 14: Search - Based on Hashtag - "#hiring"

Figure 14 shows another flavor offered by the web application which is the hashtag based search. Since some users might be interested in a specific term, for example a company or a general tag, this feature will come in handy during such situations. The figure below shows the result based on the hashtag “hiring”. This is achieved by making use of the Hashtag API as mentioned in Table 3.



USER PROFILE PAGE

Home Page

Profile Detail

Remote Jobs #OpenSalaries

Riviera State 32/106

About me

According to a research study, there are four categories of social media users: Professionals, Sharers, Creators and Bonders. These groups were formed based on similar digital footprints that they leave behind.

38 Posts

0 Friends

37475 Followers

Not Verified User

User Tweets

Remote Jobs #OpenSalaries
Nov 8, 2021 - 10:48 PM

@doximity is hiring a Remote Data Analytics Manager (~$140k, United States) #remotework #remotejobs #wfh https://t.co/KQfGxpCtcB

0 0 0

HashTags

remotework # remotejobs # wfh

Remote Jobs #OpenSalaries
Nov 8, 2021 - 2:24 AM

M-KOPA is now hiring for a Remote Senior Software Quality Assurance Analyst (Lagos) #remotework #remotejobs #wfh https://t.co/WiYcyLr492

0 0 0

HashTags

remotework # remotejobs # wfh

Remote Jobs #OpenSalaries
Nov 3, 2021 - 5:48 AM

.@lvhglobal is now hiring! Looking for a Remote Hospitality Maid (Los Angeles) #remotework #remotejobs #wfh https://t.co/kg8jI8A3Y3

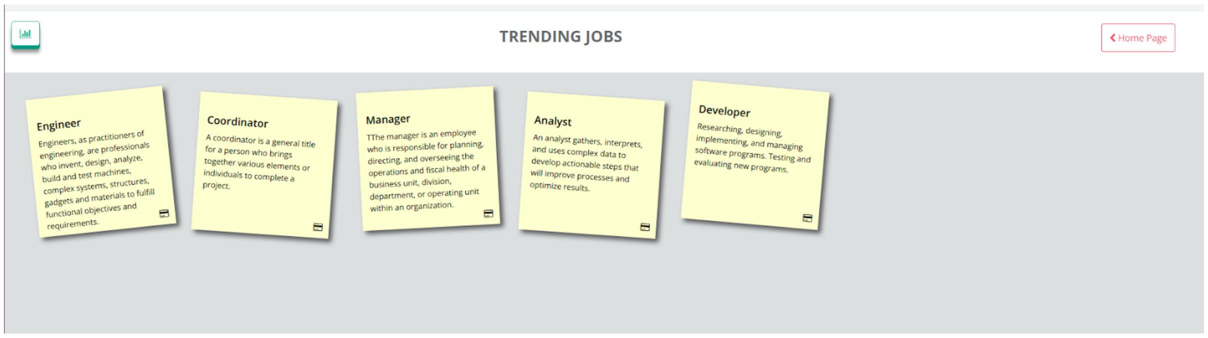
0 0 0

HashTags

remotework # remotejobs # wfh

Figure 15: Search - Based on User - User details along with Tweets

Figure 15 shows the user profile page. This will serve as the recruiters profile where the user can view all the job listings posted by a recruiter. This functionality is similar to the user profile page provided by Twitter. The User API discussed in Table 3 is used to achieve the aforementioned functionalities.



TRENDING JOBS

Home Page

Engineer
Engineers, as practitioners of engineering, are professionals who invent, design, analyze, build and test machines, complex systems, structures, gadgets and materials to fulfill functional objectives and requirements.

Coordinator
A coordinator is a general title for a person who brings together various elements or individuals to complete a project.

Manager
TThe manager is an employee who is responsible for planning, directing, and overseeing the operations and fiscal health of a business unit, division, department, or operating unit within an organization.

Analyst
An analyst gathers, interprets, and uses complex data to develop actionable steps that will improve processes and optimize results.

Developer
Researching, designing, implementing, and managing software programs. Testing and evaluating new programs.

Figure 16: Trending Jobs - Detail View

Figure 16 depicts the trending page where the users can view all the trending jobs and the top listing in them. This information is generated with the word cloud discussed in section 3.4.1. Currently this feature is static and can be made dynamic based on the input job listing batches by creating an end-to-end trending pipeline.

6 CONCLUSION

In this project, we have successfully built a scalable and fault tolerant job search platform that aids potential job seekers to find and land their dream job. In order to achieve this, we exploited Big Data technologies such as Tweet Streaming, Apache Spark Batch Processing and Apache Solr Distributed Indexing. Finally, relevant job listings were exposed as light weight APIs to the Angular web application that can be used by job seekers. Performance evaluations done on Spark framework shows sublinear increase in execution time with linear increase in problem size. This validates that the proposed system is scalable.

The presence of multiple components opens up scope for future work that can branch out in many directions. Some of them are as follows:

- Automating the entire pipeline using frameworks such as Apache Kafka or Apache Airflow.
- Expand to use distributed processing engines such as Apache Spark in the data collection phase to make it more efficient and faster.
- We currently use a library as a predicting tool for OCR. To have more control on the quality of text extracted from images we can build OCR models from scratch or use transfer learning and using dataset from job related domains that would help to increase the accuracy of text extraction.
- Using Named entity recognition for extracting location in a Tweet's text. This can help in providing a more personalized experience to users whose queries are based on location.
- Topic modelling (or Domain Tagging) on Tweets can be used to improve the retrieval metrics and also offer key domain based time-series visualization opportunities.

REFERENCES

[1] Sentimental Analysis of Twitter Data for Job Opportunities:
https://www.irjet.net/archives/V6/i11/IRJET-V6I11283.pdf

[2] Matching Recruiters and Jobseekers on Twitter:
https://ieeexplore.ieee.org/abstract/document/9381392

[3] A hashtag recommendation system for twitter data streams:
https://link.springer.com/content/pdf/10.1186/s40649-016-0028-9.pdf

[4] Optical Character Recognition by Open Source OCR Tool Tesseract: A Case Study:
https://www.researchgate.net/publication/235956427_Optical_Character_Recognition_by_Open_source_OCR_Tool_Tesseract_A_Case_Study

[5] Real-Time Twitter Trend Analysis Using Big Data Analytics and Machine Learning Techniques:
https://www.hindawi.com/journals/wcmc/2021/3920325/

[6] Sensing Trending Topics in Twitter:
https://www.researchgate.net/publication/254257985_Sensing_Trending_Topics_in_Twitter

[7] Intern Project: Creating a Global Search using Solr | Stacks & Q's (qualtrics.com)
https://www.qualtrics.com/eng/global-search/

[8] Topic Detection and Tracking Techniques on Twitter: A Systematic Review
https://www.hindawi.com/journals/complexity/2021/8833084/

[9] Case Study: Ecommerce Search Application
https://www.happiestminds.com/whitepapers/using-apache-solr-for-ecommerce-search-applications.pdf

[10] A New Approach for Automated Job Search using Information Retrieval:
https://www.researchgate.net/publication/310832275_A_New_Approach_for_Automated_Job_Search_using_Information_Retrieval

[11] Smart job searching system based on information retrieval techniques and similarity of fuzzy parameterized sets: https://pdfs.semanticscholar.org/bea7/e9010b857b2acb802f17281192e319f25e0a.pdf

[12] Help me find a job: A graph-based approach for job recommendation at scale:
https://ieeexplore.ieee.org/document/8258088

[13] Social Media Recruitment Statistics:
https://sociallyrecruited.com/2020/02/27/social-media-recruitment-statistics-2020/

[14] Apache Solr FieldType documentation:
https://solr.apache.org/guide/8_11/field-types-included-with-solr.html

[15] Apache Solr Tokenizers documentation: https://solr.apache.org/guide/6_6/tokenizers.html

[16] Apache Solr Similarity documentation:
https://solr.apache.org/docs/6_0_1/solr-core/org/apache/solr/schema/class-use/SimilarityFactory.html

[17] Apache Solr Field Documentation:
https://solr.apache.org/guide/8_5/defining-fields.html#defining-fields

[18] Solr Documentation: https://solr.apache.org/guide/8_11/