Report

Building an Apache Solr based Search Engine for DARPA XDATA Employment Data

Chen Qian
1195187409
qianchen@usc.edu

Jixin Liu
4527496546
jixinliu@usc.edu

Kang Wang
1093294783
kangwang@usc.edu

Abstract

After the crawling and de-duplication jobs are completed, the next step of processing data usually is indexing the data and querying some information about the data.

In this article we introduce how to take those reduced JSON files and build an indexing system to send the data into the Apache Solr and create an Employment job search Engine. The indexing system will include two indexing system, one is TFIDF, the other one is linked based ranking algorithm which is inspired by PageRank.

1.Introduction

In this Assignment, we will use the reduced JSON data sets that we generated from the raw TSV File. We will process those JSON file, ingest those JSON file into OODT CAS-Filemanager and and use OODT PGE to post those JSON to Apace Solr using ETLlib. Then we will write some queries querying some information about the data sets using both linked ranking and TF/IDF ranking algorithms.

2.Results and Analysis

2.1 Develop an indexing system using Apache OODT and ETLlib

2.1.a Capture metrics about the ingestion into Solr using OODT. How long did each job take on average? What were the bottlenecks? What was the overall wall clock time for the full indexing?

When using OODT for ingestion files into Solr, it takes about 0.06 - 0.08s for ingesting a single json file. For inserting partial data set(270 thousand records), it takes about 380 minutes, for ingesting the whole data set(2.2 million records), it takes about 2900 minutes. In order to find out what the bottleneck is, we try to directly use the ETLlib to post jsons to Solr, it turned out that for each single JSON file, it only take about 0.001s. After a little digging and discussion, we thought there are 3 reasons why using OODT is much slower. First, after crawling a json file, the file manager will copy the json file into its catalog, this takes time. Second, second, after crawling a file, the crawler will trigger a workflow to invoke PGE and execute the ETLlib, the initialization of a workflow and PGE tasks takes a lot of time. We find out that when initializing the workflow, the workflow manager will try to get the configurations from several xml files(events, tasks, workflow information etc). This repetitive file reading takes lots of time.

2.2 Develop an indexing system using Apache OODT and ETLlib

2.2.a LinkedBased Ranking Performance

We choose to run the whole data set. The size of job postings is about 2 million.

At first we build graph based on the potential relationship of job postings. The mechanism is illustrated in the 3.c . The generated graph is stored as inverted index for each job rather than matrix. This method reduces the space consumption tremendously.

Generate graph takes 777513 ms in total.

Then we use multi-thread to compute ranking value with 100 thread.

Page ranking process takes 1760236ms. About 42-45 second per iteration.

In order to make the whole data set run on page rank with assign 15GB memory to JAVA. The average memory usage is about 12.5GB.

2.3 Develop a suite of queries that demonstrate answers to the following challenge questions using your indexed data and your ranking algorithms.

2.3.a Predict which geospatial areas will have which job types in the future.

We predict future job types based on the past. The basic idea is choosing most high frequency job type as the prediction of future job type of specific geospatial area.

We employed facet feature of Solr to count job types in a given geospatial area. Input are latitude and longitude, output is job type.

The query example like this:

http://localhost:8983/solr/collection1/select?q=*%3A*&fq=longitude%3A%22-74.0733387%22&fq=latitude%3A+%224.6710155%22&wt=json&facet=true&facet.query=facet.sort%3Dcount%26facet.mincount%3D1&facet.field=jobtype&rows=0

2.3.b Compare jobs in terms of quickly they're filled specifically in regards to region.

The idea of this question is computing difference between post date and last seen date. The smaller difference means this job is filled faster, so we group result by the job existing duration. Input is location and output are duration and their count.

The query example like this:

http://localhost:8983/solr/collection1/select?q=*%3A*&fq=jobLastDays%3A%5B0+TO+10000%5D&fq=location%3ACundinamarca&sort=jobLastDays+asc&rows=600&fl=jobLastDays&wt=json&indent=true&group=true&group.field=jobLastDays

2.3.c Can you classify and zone cities based on the jobs data (E.G. commercial shopping region, industrial, residential, business offices, medical, etc)?

To solve this problem, we firstly emulated seven basic categories: Commercial, Industrial, Residential, Business, Medical, Education and Service by using Solr to extract high frequency keywords in job titles. And then we mapped each job record to a specific category. In the end, we group result by category and return the top category to judge which kind of type the region belong to. Input is location and output is category.

The query example like this:

http://localhost:8983/solr/collection1/select?q=*%3A*&fq=location%3ACundinamarca&fq=!tag%3An%2Fa&fl=tag&wt=json&indent=true&group=true&group.field=tag

2.3.d What are the trends as it relates to full time vs part time employment in South America?

The idea of this question is the first-step is excluding North America countries like Mexico, Dominican Republic,etc. And the second-step is counting full-time job numbers and part-time job numbers in each recent post date. For example we choose recent one month post dates like from 2013-11-13 to 2013-12-13. If the count of full-time is greater than part-time, we can conclude the trend in South America is full-time employment.

The query example like this:

http://localhost:8983/solr/collection1/select?q=*%3A*&fq=postedDate%3A%5B2013-11-13+TO+2013-12-13%5D&sort=postedDate+asc&rows=100&fl=jobType&wt=json&indent=true&group=true&group.field=postedDate

2.4 Develop a program that can either in batch mode or interactively demonstrate answers to the questions from #3

2.3.a

```
query result:
The potenial job type in the future is Tiempo Completo
```

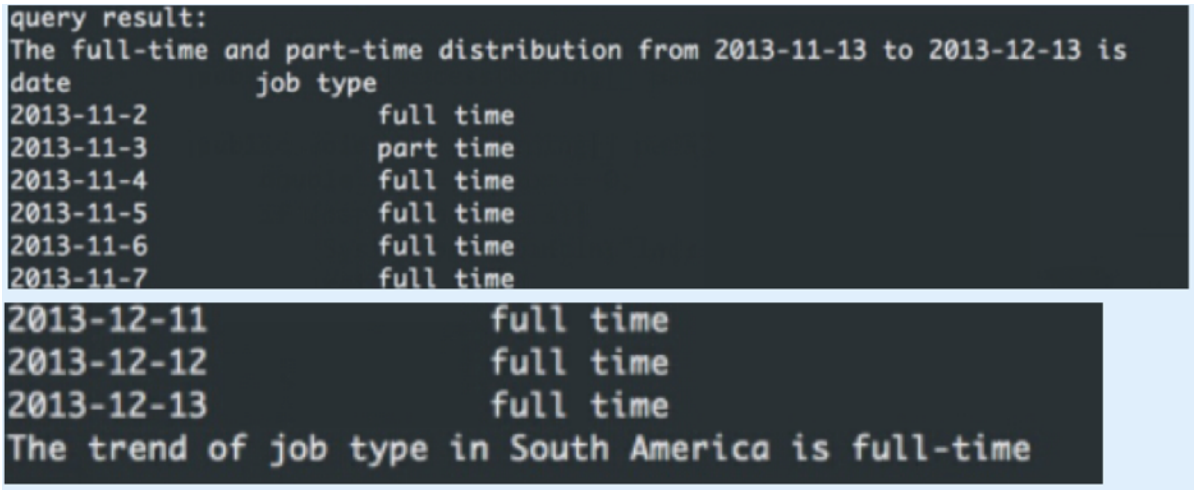
2.3.b

```
query result:
job last days           number of job
0                               449
27                              98
28                              61
29                              127
30                              322
31                              131
32                              308
33                              229
34                              195
35                              311
36                              294
37                              395
38                              6
39                              189
```

2.3.c

```
query result:
This city or zone is Educational city or zone
```

2.3.d



3.Questions and Answers

3.a how effective was the link-based algorithm, compared to the content based ranking algorithm?

Link-based algorithm is very effective in calculating document's "popularity" and designers are able to customize how to build link between each document. On the other hand, link-based algorithm is also more efficient than content based ranking algorithm, because link-based algorithm is pre-computed algorithm, in each query, for link based algorithm system just need to search ranking value, while for content based is required to do a serial of computations based on term frequency and inverse document frequency.

3.b What challenge questions were more appropriate for the link based algorithm compared to the content one?

For link based algorithm, it shows how popular the doc is regarding to user's query. However, for content based algorithm, it shows how similar the doc is regarding to query. In this case, when the query is associate with how popular or how important the job, link based algorithm will be more competitive. Since this attributes can be retrieved and evaluated through the ranking value.

The query can be various. Such as the top ten popular jobs in Columbia. Or which the trends of need of one kind of jobs in Argentina.

3.c What do you think were the strengths of OODT? What did you think were the weaknesses of OODT?

Apache OODT is a grid middleware framework for science data processing, information integration, and retrieval. OODT is used on a number of successful projects at NASA's Jet Propulsion Laboratory/California Institute of Technology, and many other research institutions and universities.

From my experience with OODT ingesting the JSON files, I think OODT is a very powerful tool for data processing. The OODT's components that I used in the assignment are Crawler, File Manager, WorkFlow and PGE. The reason that I think the OODT is powerful is because that the combination usage of different components makes OODT very flexible and can handle various types of different tasks. Moreover, each component can be highly customized. For example, when using the crawler, the crawler can crawl not only a single file, but can also crawl directories. When crawling a file or folder, you can use lots of different types of metadata extractors to generate metadata, you can use extractors provided by Tika, or even your own customized extractors that can extract whatever metadata you want. You can define an action that the crawler can trigger after finishing a crawling task including invoking a workflow, doing some actions in file manager or even sending a notification email. If you choose to invoke a workflow, there are lots of more task you can do. With highly flexible components combination

and highly customized component itself, the OODT is really a powerful and flexible data processing and information retrieving tool. What makes OODT more powerful is that OODT is open sourced, which brings lots of potential in the future.

However, though OODT is powerful, it is not perfect. There are still lots of parts that OODT can be improved. For example, the documentations for OODT are really poor. OODT can do lots of complex tasks, but it is a pain for novice users to understand how different parts of OODT work together, what files need to be configured, how to configure those files, what environment variables needs to be set but there are not many examples comparing with Solr. You have to configure all components and during this assignment, we spent a lot of time figuring out those problems.

Moreover, using OODT to repetitively invoke workflow and PGE takes lots of time. There should be some way to avoid repeatedly reading configuration file for the same type of operation. This overhead is very apparent when doing operation on small sized files.

In a word, OODT is powerful, flexible but experienced users oriented not friendly for new users, also there should be some optimization for some repetitive operations.

3.d Describe in detail and formally both of your ranking algorithms. You should describe the input, what your algorithms do to compute a rank, how to test them (and prove that they are working as expected).

For content based ranking algorithm, we chose term frequency and inverse document frequency to compute ranking and Apache Solr integrates inverted indexing functionality. And In function query of Solr, it supports a bunch of relevance functions like docfreq(), termfreq(), idf(), tf(), norm() and so on. Besides, we also boost the query by assign important attributes with high weight such as company and title. So for using this feature of Solr, we customized the configuration of Solr to add our own json file schema into schema.xml and used Apache OODT to push job postings data into Solr. After data ingestion, we inputted some queries to test precision of inverted indexing. On the other hand, we also used this content based ranking to extract high frequency keywords from job title for emulating job categories are used in Task 3.

For link based ranking algorithm, we employed Google page ranking algorithm. First, we picked four columns: company, title, longitude and latitude as key attributes to build link between each job posting record. And we integrated longitude and latitude as one key attribute geolocation, if any two job posting records have two or more common key attributes out of three(company, title and geolocation) we build a link between these two records. The reason we choose these columns as key attributes is that these attributes are widely distributed rather than job type and start which only have limited value. Choose 2 of the 3 is also satisfy the target, make the key widely distributed. In this case, it avoid the situation that large amount of jobs link to each other since they have the same key, the chosen attributes.

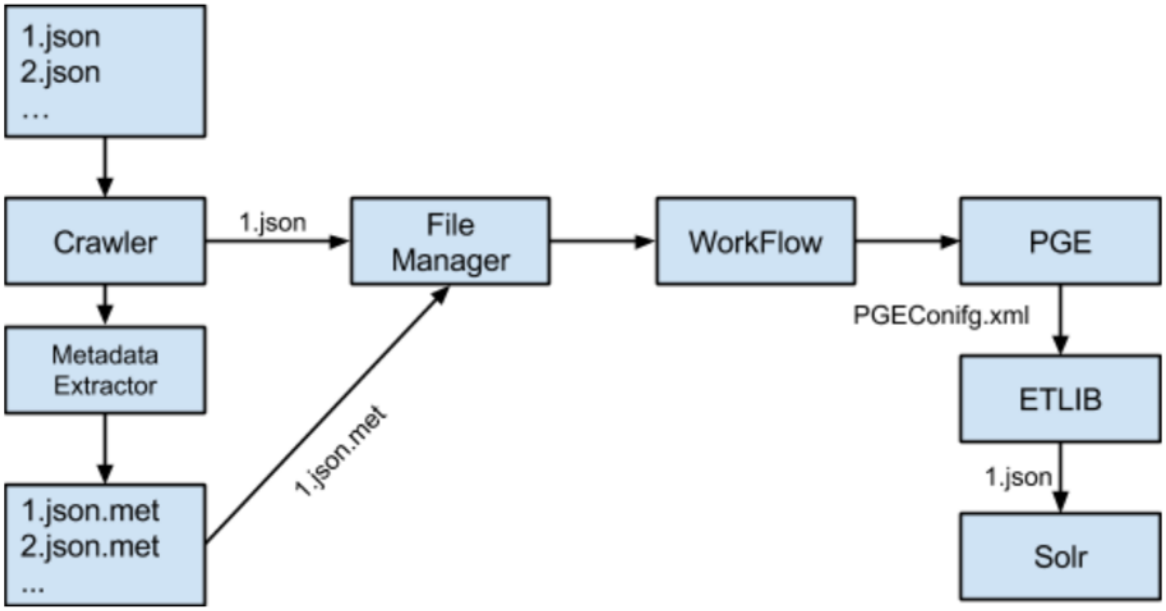
After generating the graph of job postings, we used following formula to calculate page ranking value:

$$PR(A) = (1-d) + d\ (PR(T1)/C(T1) + ... + PR(Tn)/C(Tn))$$

we set d = 0.85 and ran PageRank algorithm 40 times to get final PageRank value, and for improving efficiency we opened 100 threads for PageRank computation by using Java thread pool executor. For testing our link based algorithm, we firstly used content based algorithm to group records by location and company that sorted by frequency and then we select first 1000 records by page ranking value in decreasing order and also group them by location and company, in the end we compared these two results and checked some top groups' relative order in each result, we found the two results are basically the same.

3.e Describe the indexing process – what does your OODT workflow do? How does it interact with ETLlib?

The indexing process consists of 4 major parts: crawling(metadata extracting), ingesting, workflow and PGE(ETLlib invoking).



Crawling & Ingesting

The crawler will crawl the json folder, the metadata extractor we used is Tika extractor. After generating the metadata, the crawler will invoke the file manager, the file manager will ingest the json file and its metadata. Then the crawler will trigger a Workflow Manager action with event name GenericFileIngest that defined in Workflow's events.xml.

WorkFlow & PGE

After the WorkFlow with event name GenericFileIngest is invoked, the task that belongs to the GenericFileIngest event will be executed and a PGE task that defined in the PGEConfig.xml will be invoked. The PGE task will executed the ETLlib command line that is declared in the <cmd> tag of the PGEConfig.xml, and post the json that crawled by crawler to Solr. The file name argument of ETLlib's poster command is provided by the metadata that generated by the Tika extractor.

3.f Also include your thoughts about ETLlib – what was easy about using it? What wasn't?

The ETLlib poster function did really good job for posting JSON to data. It fits our need perfectly, generally speaking, in this assignment, ETLlib works really well.