CITY UNIVERSITY OF HONG KONG

TripAdvisor Dataset Processing

- Quality Ratings and Comments Semantics Analysis -

CS 4480
Data Intensive Computing

Created By:

LIUNARDO Mikhael	53647374
LAO Choi Hin	54045244
SANJEEV Karan	53682850
BICHER EL OUDGHIRI Omar	40101040
GUERIN Vincent	40101026

November, 2016

Course Leader: Dr. Sarana Nutanong（陳伊）

I. Introduction

Data processing on large scale dataset is known to be challenging. The concern of memory management, computational resources, time complexity and data structure management are some of the key issues that needs to be dealt as the size of our dataset increases. The need for strategic framework and techniques to process the large dataset has led and motivate us to develop this project; Tripadvisor Data Processing.

Tripadvisor is an American reviewing company which gives reviews to hotels and restaurants. The Tripadvisor dataset is a collection of reviews given by customers on various hotels in the United States of America. The attributes of each reviews varies from the cleanliness, room service, overall quality, costs and other aspects of rating scoring system. Each reviewer would also include their comments into the reviews, expressing their opinions towards the hotel.

Tripadvisor data processing project is aimed to develop a program framework which is able to perform quality ratings and comment semantics analysis on the large size dataset of tripadvisor. The developed framework will process the raw data into a structured format and analysis is performed on the dataset to extract "interesting" and "meaningful" knowledge from the data collection.

The framework or technical approach that we use to process the Tripadvisor dataset is Hadoop Distributed File System (HDFS). The performance of Hadoop applications is sensitive to all the components of the stack including application code, HDFS, JVM, OS, Network and hardware's quantity and computational power. The Tripadvisor projects emphasizes on 2 major field of concentration; developing MapReduce application code to process raw data into computed structured data and Pig Latin queries design to extract relevant information.

The development of MapReduce application code will be done in Java Programming Language and queries application will be done in Hadoop Pig Latin platform. The data processing will be done in a cluster environment (a collection of nodes or machines operating synchronously to process dataset).

II. Related Work

There are many research and projects done by companies all around the world which attempts to process text data and extract relevant information from the data collection. In most cases, these text data are associated with some numerical data which can be quantified together to find association or trends within the particular field or industry.

Each research utilizes different techniques and algorithms to process these data. For instance, Google is known to infusing NLP (Natural Language Processing) in its search engine which processes the textual content of the extremely large collection of web pages it indexed. The company also uses its own ranking system (PageRank is one of the simplest form of ranking based on redirection count, Google have its own confidential algorithm) to quantify the processed textual data into ranks of web pages.

The Tripadvisor project attempts to create our own framework to process large data set and produce "meaningful" data extracted from the raw dataset. The dataset for this project is comparatively small compared to the tech-giant Google, however we are doing comparative analysis on different approaches throughout the experiment to broaden the experimental result on different potential techniques to be adopted in larger scale projects.

III. Methodology

A. Dataset Analysis

The Tripadvisor dataset used in this project has a size of 2.3 Gigabytes with 11865 files of hotel reviews in total. Each file corresponds to a collection of reviews for a single hotel. Each file is stored in JSON file format containing many review objects; each review objects has several attributes: cleanliness, overall score, front desk service, sleeping quality, room quality, money value, business service, room service, location, author, comments and date of comment.

The review object in JSON data structure has 2 different data types; numerical and textual. The numerical information are ones coming from cleanliness, overall score, front desk service, sleeping quality, room quality, money value, business service, room service and location ratings. The textual data (text types) originates from the comments and author attributes.

B. Hadoop Distributed Computing

Hadoop distributed computing is a framework which processes data in multiple nodes. Each nodes or machine has their own data storage and processing units. The idea of Hadoop distributed computing is to optimize local computation of each nodes by exploiting the local usage of the data storage. Data communication between nodes or machines is only performed when it is necessary (minimize communication penalty).

There are two types of nodes in Hadoop distributed computing; master and worker nodes. The master nodes is responsible to maintain an index of the mapping between worker

nodes and the data storage it has, thus it can exploits local computation of each nodes. The master node can be seen as a "scheduler" for jobs distribution and the worker nodes are responsible for performing the necessary processing and computation.

C. Semantics Analysis Technique

Semantics analysis is a technique which attempts to process texts information in a more quantifiable and justifiable approach. Understanding text information through lexicography is difficult as program codes are not compatible with human language. However converting text information into numerical data representation allows the program code to process texts in a quantifiable approach.

The Tripadvisor project attempts to process these texts information into Term Frequency - Inverse Document Frequency (TFIDF) numerical scores. Term Frequency - Inverse Document Frequency is a numerical statistic that is intended to reflect how important a word is to a document in a collection of corpus. The TFIDF score of each word can help us to understand the importance of a word and its associative rating scores given in the collection of reviews.

Computing TFIDF for each word is a difficult task without a proper approach and algorithm. MapReduce is the chosen technique to compute the TFIDF for each word and different approaches is used to develop and determine the best program code design.

D. Structured Query Language to Pig Latin Query

SQL (Structured Query Language) is a declarative programming language used in relational databases. Unfortunately, in the case of Tripadvisor data processing project, the data set are not structured; the data are not displayed in a relations and we do not intend to form it into relational database due to the sparsity of the data and the analytical text aimed to be done on each word level. Therefore, SQL is not applicable in this case and led us to use Pig Latin Query instead.

The Pig Latin Queries is suitable for the need of Tripadvisor project, given that the queries can support flexible or semi-structured data, we can easily manipulate the records to find interesting and meaningful trends on the dataset. The Pig Latin queries is similar to SQL to a certain degree; it supports the major native feature of SQL including JOIN, FILTER, CROSS, GROUP BY and other forms of record processing features.

IV. Experiments

A. Dataset Preprocessing

Dataset preprocessing is required to be done before the actual MapReduce jobs is executed. Given that the original dataset are stored in JSON format, conversion to simple plain text file (.txt files) for the comments data needs to be done firstly. The numerical data are also processed into comma separated values (.csv files). The purpose of changing the format of the data is to ease the process of loading the data into both Pig Latin and the Map Reduce jobs.

Specific for textual data derived from the comments data attribute, further data preprocessing is required. Eliminating punctuations and stopwords (meaningless words; a, the, he, she, it, then, are, was, etc.) is performed during the textual information extraction. The purpose of performing punctuation and stopwords elimination is to further enhance the data quality that we will obtain in the TFIDF computation process. Words that have no meaning and are to be disposed and disregarded in the TFIDF computation.

B. TFIDF MapReduce Processing

The TFIDF computation for each words in the comment section is done through MapReduce jobs approach. The basic idea of the approach is to perform a certain processing, applied on each token/word level in the map phase and aggregate the results from map phase into the reduce phase to obtain specific computational result.

The TFIDF formula is shown in the figure below.

$$w_{i,j} = tf_{i,j} \times \log\left(\frac{N}{df_i}\right)$$

$tf_{i,j}$ = number of occurrences of i in j
df_i = number of documents containing i
N = total number of documents

Before the TFIDF can be computed, for each word we need the numerical value of the word count, number of word in the respective document and the number of times the word appear in all documents. The MapReduce jobs is aimed to provide the required information for each word TFIDF computation.

In simple explanation, the MapReduce framework can be explained in the following:

1. Map step: Each worker node applies the "map()" function to the local data, and writes the output to a temporary storage. A master node ensures that only one copy of input data is processed.
2. Shuffle step: Worker nodes redistribute data based on the output keys (produced by the "map()" function), such that all data belonging to one key is located on the same worker node.
3. Reduce step: Worker nodes now process each group of output data, per key, in parallel.

There are 4 main jobs of MapReduce in the TFIDF computation. Each jobs is specific in computing a certain numerical value; (i) Word Count, (ii) Occurrence Count, (iii) Document Occurrence Count, (iv) TFIDF calculation. Each jobs will results in a key-value type data structure with a calculated numerical value as the “value” result.

In this Tripadvisor project, we utilizes 3 different approach to perform the MapReduce TFIDF calculation and records the performance for each approach.

1. Naive Approach (Java HashMap)

The Naive approach in performing the TFIDF MapReduce jobs does not attempt to utilize cluster or Hadoop as its basis for computation. Simple Java program code was developed to store tokens/words in HashMap data structure.

We started the naive solution with class modeling in UML. Then, we start implementation in java with eclipse. We have created 3 class, class hotel, review and main.The hotel class consisting in list of reviews and we have created a hash map of hotels (the key was hotel Id and the value an hotel object) in the main.

We also have created a second hash map (the key was hotel id and the value a hash map where the key was a word and a value was frequency of this word in comments about this hotel).

Given that the comments dataset is very large (2.1 Gigabytes), the process of constructing the complete HashMap containing all the dictionary item took a very long time. As the program runs extremely slow,, we deem this method as ineffective to manage this scale of dataset and decided not to pursue the completion of all jobs. The HashMap approach only managed to perform the Map step in the 1st Job and it consumes 2 hours to perform this single task but we didn't get a memory error. The aim of this naive solution was to show that java

program is not time efficient and we should use Hadoop framework because of its distributed architecture.

2. MapReduce Jobs Processing

The MapReduce Jobs processing approach utilizes Hadoop framework as its basis and performs the data processing on clusters (collection of nodes or machines). The MapReduce is developed in Java code consisting of 4 classes. Each class is responsible for each one of the 4 jobs needed to be computed to obtain the final TFIDF result.

In arbitrary naming scheme, class A, B, C and D are the classes used in this Tripadvisor project. Each class contains 2 functions which are the Mapper() and Reducer(). The Mapper() function is executed locally in each worker nodes by exploiting the local data storage and after shuffling is performed, the Reducer() function will aggregate the result to obtain the specific data.

In simple terms, the Mapper and Reducer for each jobs can be explained as below:

a. **Word Count:**
    i. Mapper: Tokenize the text files; each word is taken into tokens and is written in key-value format. Key as the tokens and value as 1 numeric
    ii. Reducer: Aggregates the token together to compute the count of each word by aggregating the 1's numeric
b. **Occurrence Count:**
    i. Mapper: The occurrence count is actually meant to calculate the total words appearing in the document; number of word occurring in a single document. The mapper produce key of documents and word-count as value.
    ii. Reducer: Aggregates the output from the Mapper step to compute the total words in a document
c. **Document Occurrence Count:**
    i. Mapper: Map the data based on the document and word as a key and add 1's numeric into the associated value
    ii. Reducer: Aggregate output from Mapper() step to sum all the 1's and return the result for the number of documents where the word occurs
d. **TFIDF Calculation:**
    i. Mapper: The mapper computes the TFIDF value for each word and produce key-value pair of word and TFIDF value.

ii. Reducer: The reducer is only an identity function in this last job

3. Optimized MapReduce Jobs Processing

The Optimized MapReduce Jobs holds the same logic as the Non-optimized MapReduce jobs in the previous section. The only difference is that in the Optimized version, memory hacking is done in order to minimize data communication penalty.

Communication among nodes is expensive can be result in increase of the entire program execution time. The purpose of the optimization is to minimize data shuffling volume by performing LOCAL AGGREGATION. Local aggregation means that we aggregate the values of the same keys in the local or mapper() step. In this project, the local aggregation is done by creating another data structure (ArrayList or HashMap based on conditions and job's nature) in the Mapper() function. The HashMap or ArrayList data structure is used to temporarily store the data and aggregation is done inside the Mapper() function with the help of these data structure.

Note that the aggregation using helping data structure (HashMap or ArrayList) is done in each worker node. As a result, exploitation of local aggregation is achieved and the volume of the data communicated in the shuffling process can potentially be diminished.

C. Pig Latin Queries

After the data cleanup and TFIDF calculation, our dataset is ready for deeper analysis. We have developed a few queries in Pig and see if we can extract some of the useful information.

1. Controversy (Score's standard deviation)

In TripAdvisor, reviewers are not expected to be professional hotel judges with standardized scoring criteria. A general public may give his opinion according to his only point of view. In that sense, biases are created and most of the time an average score cannot directly tell the performance of the hotel with such extremities. For example, a controversial hotel may receive a lot of 1 mark and 5 marks at the same time, resulting a average score of 3, but that may be mistakenly inferred as the hotel did a so-so job for every customer.

To avoid such misconception, new information must be provided to generalize the stability of service. We have chosen standard deviation for that purpose. Standard

deviation in a set of numerical data is a statistical number to measure how each data varies with the average of that set of data. The higher the number, the more diverse the data set is. Below is the equation of calculating standard deviation:

$$s = \sqrt{\frac{\sum(x - \bar{x})^2}{n - 1}}$$

In the formula, each data x is subtracted by the mean of the data $\bar{x}$, each of the differences are then squared and summed up together. After that, the total is divided by the number of data n - 1, to form the variance, in which the square root will be the standard deviation.

For our dataset, we calculated the standard deviation on the selected performance metrics for each hotel. To be more specific, the metrics are cleanliness, front desk, sleep, quality, room service and location. The reason to choose these attributes due to their availability among the reviews, as reviewers were not required give score on every aspect and these attributes were very likely to be scored.

At first we need to get the average and the record counts of each metrics for each hotel, to enhance the performance, we filter out the reviews that are all null on those attributes and then just project the needed columns for calculation. The next step is just compute the average and the count. After that, each metric on each record used to calculate the squared difference with its corresponding average and hotel ID. At last, the standard deviations are computed by summing up each attributes, and divided by the respective count - 1.

2. Contradiction (Irrational review detection)

As said before, reviewers on TripAdvisor could be anyone around the world. With such broad user base, there may be cases that these critics will give opinions that does not make any sense. Contradiction is one of the example, in which users may say something negative in their textual comment, but eventually give a high score on the things that they are not satisfied. Such reviews may considered as “contradictory” as they were irrational.

To cater such problem, we have developed a pig script to find the irrational comments on each hotel based on a particular score attribute and TFIDF of the attribute’s related

word. In our case, we will use cleanliness and the word “dirty”. The reason to choose this word is that “dirty” is rarely used in negation (“not dirty”).

Overall, the program involves two tables, the score and TFIDF. To get the result, join operation must be used. However, knowing that both of the tables are large in scale, filtering and projection must be done before the join in order to preserve efficiency. For the score table, we first project the hotel ID and compute average cleanliness score of each hotel, then we filter the hotel which have the score over a certain threshold. For the TFIDF table, we keep the records which contain the word “dirty”. After that, we further filter out the records which do not satisfy the condition of 0.1*MAX(dirty_TFIDF) to make sure the word is commonly used among the hotel. When these processes are finished, the join process will provide our desired result, since a successful join means that there is a hotel where people give high score in cleanliness but use the word “dirty” often in the comment.

3. Correlations

We have created Pig queries to know more about top words correlation with good or bad grade.
First of all, we have created query to know the top 300 words (about all the hotels) and we have found that words had generally positive meaning and were mostly in English, French or Spanish languages.

Then, we asked ourselves if some if best grade from overall criteria were legit or not so we did a query choosing top 50 hotels (overall criteria) and their associated grade for overall criteria. They had all 5.0 grade so it confirmed our intuition that some hotels had boosted reviews. We decided to slowly add more than 50 and we have a relevant number at 300 :
- 160 were fully boosted and at 5.0 grade.
- 140 were a bit boosted and with a grade between 4.8 and 5.0.

Corresponding query :

```
generateAverageOverall = FOREACH reviewGrp GENERATE group,
AVG(review.overall) as averageOverall;
sortByOverall = ORDER generateAverageOverall BY averageOverall DESC;
outputRequest6 = LIMIT sortByOverall 300;
```

Now that we knew more about both tables separately, we have chosen to join them in order to understand which top n words were associated with top n best or worst hotels on specific criteria.

Our methodology was the following :
- Generate average on selected criteria.
- Sort hotels on that average and select 500 hotels with LIMIT operator.
- Make a natural join of this results with tfidf table on Hotel Key.
- Group that table on words and show top 200 words.

We have first applied that methodology to get top 200 words for top 500 hotels on overall criteria. We wanted to know which words were associated with the fakes grades that we have found earlier. We discovered that it was mostly German and Spanish words so it could suggest that boosted hotels got reviews in those languages.

But we thought that if top reviews were fake and bottom reviews might be more honest and strongly associated with bad meaning words. So we first have chosen to analyse top 200 words for 500 worst hotels on cleanliness criteria. As a result, our intuition was confirmed and we have found bad meaning words (especially in French) so it seemed that bad reviews were more strongly correlated with words. For example, in top 10 words, propreté and douteuse that gives propreté douteuse which means bad cleaning in French.

We kept going with our analysis with top 200 words for 500 worst hotels on factors of business service grade and room service grade. Our intuition was not confirmed as we have found some words (sympa --> nice, charme --> beautiful but not the negation pas --> not) that were correlated with good meaning. Of course, most of the words found were correlated with bad meaning but we found it interesting to notice it. As a conclusion, we can say that a correlation exists with grade and words but we can find some anomalies. Also the average on criteria allows some hotels with really bad reviews to get counterbalanced with few really good reviews and mix good and bad reviews (of course, one side composed the majority of the hotel so its grade is either really good or really bad). As another analysis, we can think of selecting only comments that are below or above a certain grade in order to determine which words are associated with really good or bad grade (but we will lose the average on a certain criteria).

4. Pig Latin Optimization

Pig Latin allows user to control most of the optimization due to lack of cost based optimization built into Pig optimizer. The people who developed Pig Latin have implemented different operators for different situations (Example: Replicated Join, Merge Join, COGROUP) and allow the user to choose which to implement in any

given pig query. Hence, empowering the user allowing more control but at the same time increasing the burden on the user. This specific concept is in coherence with the design principles laid down by the Pig Latin developers which is "Pig is a domestic animal", which means it does whatever the user commands it to do.

In the following section, we briefly discuss the rule based optimization techniques used in our project:

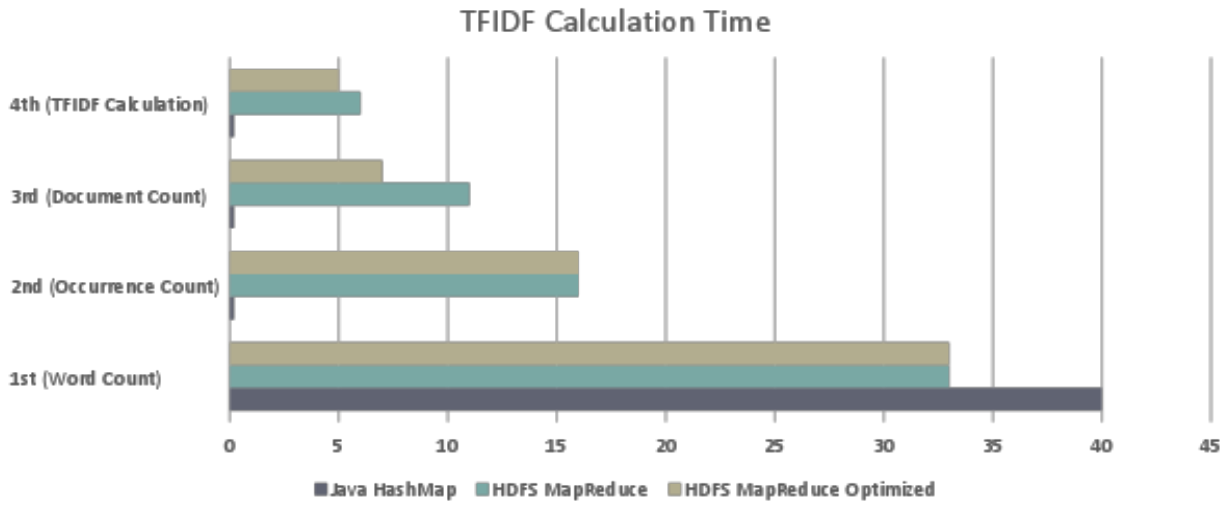
a. Applying filter as early as possible – Push-up Filter.
b. Applying flatten produces a cross product as late as possible in the plan, keeping the number of records low in the pipeline – Push-down Flatten.
c. Omitting columns never used or will no longer be used, reducing the size of records – Column Pruner.
d. Omitting map keys never used, again reducing the size of data by filtering out records that are NULL – Map Key Pruner.
e. Limit and reduce the number of records – Limit Optimizer.
f. Forcing certain operations to take place in memory and therefore, resulting in a performance boost – Specialized Joins.

Before and after applying the above optimization rules, we performed various experiments and recorded their performance.

V. Results

A. MapReduce Job Results

The result of the MapReduce jobs using 3 different approaches is displayed in the following charts:



Based on the data collected, we can deduce that the performance of MapReduce jobs with local aggregation exploitation gave the best performance in terms of time complexity.

MapReduce using Naive approach of HashMap data structure has proven to be ineffective. Due to the extremely slow time consumption and Memory Overflow error thrown, it safe to conclude that utilization of ANY data structure without proper memory and computational management will result in failure, especially when the dataset size is large.

The analysis on Hadoop based MapReduce jobs; both optimized and non-optimized, has led to the deduction that MapReduce jobs with local aggregation produces a better result in terms of time performance. This is result aligns with our reasoning and stance that data communication is expensive and local aggregation and reduce the penalty of data communication in the shuffling stage.

Another interesting finding is the fact that only 2 jobs out of the 4 MapReduce jobs got affected by the optimization process. Job1 and Job2 (Word Count and Occurrence Count) did not experience any time improvement although local aggregation has been performed. Our analysis has led us to a deduction that Job1 and Job2 were not affected by the optimization due to the fact that there is NO or MINIMAL local aggregation tasks that can be done in the local node. Both Word Count and Occurrence Count job nature does not have significant local aggregation "quantity" to be done in the local nodes, thus either using optimized or non-optimized will not led to any significant change in the performance result. Comparatively with Job3 (Document Occurrence Count), there are many "quantity" of local aggregation that can be done in this job which led to the time reduction of the entire job computation time.

B. Pig Latin Queries

1. Controversy (Score's standard deviation)

To have a better focus on the controversy, we only projected the hotels that have standard deviation over 2.5 in any aspect. As a result, a total 1427 records were emitted. In general, most of standard deviations that match the last projection filter are in a range about 2.5 - 2.7. And, interestingly, these debatable hotels in overall are likely to be controversial on every aspects. In other words, there are rare cases that a hotel only get one attribute with high standard deviation. This may concluded that, for each customer, hotels tend to provide the same set of quality of services.

In terms of performance, the whole Pig script took around 10 minutes to run on the CSLab cluster.

2. Contradiction (Irrational review detection)

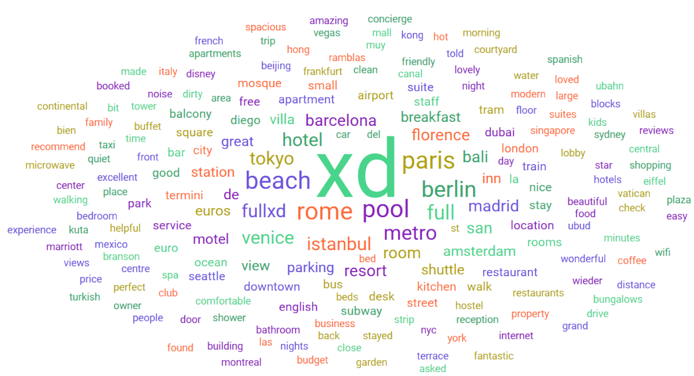
Initially, we have set the threshold of being high cleanliness score is 4. But eventually, after all the jobs finished running, no record were emitted. Having checked that both score and TFIDF tables contain records after filtering, it is concluded that there are no successful join between the two relations. However, to make sure the Pig script is correctly designed, we have lowered the cleanliness threshold to 3. Finally, one record was successfully joined.

For this hotel, the average cleanliness score is exactly equal to 3, with a TFIDF of the word “dirty” around 0.016. With such limited output, we can conclude that, in terms of cleanliness, there were no irrational commenters to pollute the data and viewers may give better trusts on those comments.

In terms of performance, the whole Pig script took around 20 minutes to run on the CSLab cluster.

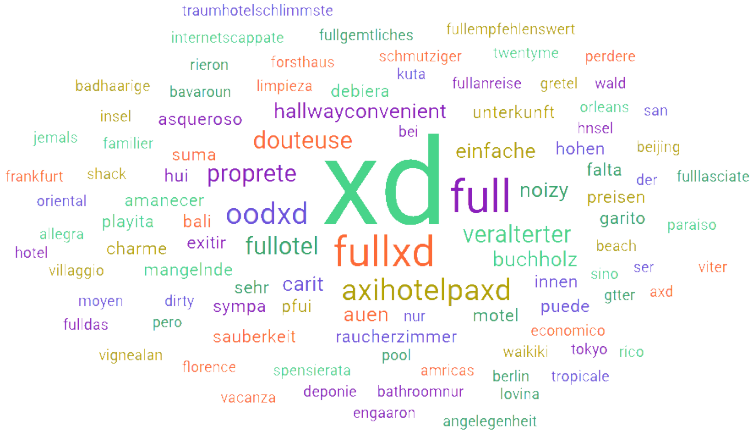
3. Correlations

a. Top 200 words



b. Top Cleanliness

c. Worst Cleanliness



d. Worst Business and Room Services

4. Pig Latin Optimization

A summary of the results of Pig Latin optimization experiments is as follows:

a. After “Filter Early, Project Early and Remove NULL values”, we observed that there is not necessarily an optimization in time by applying this optimization

rule but we definitely note a space and resource optimization. When we executed different queries, we observed that non-optimized queries occupied more cluster space and would go over the "fair usage" limit when compared to the optimized queries in the Hadoop Web Interface.

b. Pig Latin does column pruning automatically when a schema is defined and hence, we did not note an optimization in time with the queries. Therefore, when we try filtering out columns, there is an I/O cost as the columns are pruned manually.

c. The simple JOIN vs replicated JOIN showed that the latter uses the RAM to join a smaller table to a bigger table and therefore, we observed a significant optimization in time. When we used all the resources available to cluster, we observed a massive upgrade in performance. We not only noticed that CO GROUP outperforms simple JOIN and replicated JOIN, but also that Merge JOIN and Sparse Merge JOIN work only with a two-way join.

VI. Conclusion

Through experimentation of different approaches used in this Tripadvisor project, some key observations has been made which led to immaculate conclusions of MapReduce and Pig Latin Queries job performance. The key conclusions that can be drawn from the project includes;

1. Performing large scale data processing without proper memory and computational framework or strategy will lead to failure; relying on good selection of data structure is not sufficient to manage the large dataset
2. MapReduce jobs on distributed computing cluster framework is proven to be capable of managing large dataset. Given that the cluster contains sufficient nodes for the entire process to run in a effective manner
3. Optimization of MapReduce jobs; particularly on exploitation of local aggregation, are only effective on jobs where local aggregation can be applied for the major components of the mapped key-value pair. Otherwise, the optimization approach will not cause any significant performance improvement.
4. Pig Latin optimization is an iterative process and needs to be done continuously. Techniques used for optimization depends on the data or query and performance may improve when hash aggregation is used. When we directly translate from SQL to Pig Latin, it always results in a non-optimized query.

Appendix

SQL FUNCTION	SQL	PIG
SELECT	SELECT column_name,column_name FROM table_name;	FOREACH alias GENERATE column_name, column_name;
SELECT *	SELECT * FROM table_name;	FOREACH alias GENERATE *;
DISTINCT	SELECT DISTINCT column_name,column_name FROM table_name;	DISTINCT(FOREACH alias GENERATE column_name, column_name);
WHERE	SELECT column_name,column_name FROM table_name WHERE column_name operator value;	FOREACH (FILTER alias BY column_name operator value) GENERATE column_name, column_name;
AND/OR	... WHERE (column_name operator value1 AND column_name operator value2) OR column_name operator value3;	FILTER alias BY (column_name operator value1 AND column_name operator value2) OR column_name operator value3;
ORDER BY	... ORDER BY column_name ASC|DESC, column_name ASC|DESC;	ORDER alias BY column_name ASC|DESC, column_name ASC|DESC;
TOP/LIMIT	SELECT TOP number column_name FROM table_name ORDER BY column_name ASC|DESC; SELECT column_name FROM table_name ORDER BY column_name ASC|DESC LIMIT number;	FOREACH (GROUP alias BY column_name) GENERATE LIMIT alias number; TOP(number, column_index, alias);
GROUP BY	SELECT function(column_name) FROM table GROUP BY column_name;	FOREACH (GROUP alias BY column_name) GENERATE function(alias.column_name);
LIKE	... WHERE column_name LIKE pattern;	FILTER alias BY REGEX_EXTRACT(column_name, pattern, 1) IS NOT NULL;
IN	... WHERE column_name IN (value1,value2,...);	FILTER alias BY column_name IN (value1, value2,...);
JOIN	SELECT column_name(s) FROM table1 JOIN table2 ON table1.column_name=table2.column_name;	FOREACH (JOIN alias1 BY column_name, alias2 BY column_name) GENERATE column_name(s);

Figure 1 Rule Set 1 for Query Conversion

SQL FUNCTION	SQL	PIG
LEFT/RIGHT/FULL OUTER JOIN	SELECT column_name(s) FROM table1 LEFT|RIGHT|FULL OUTER JOIN table2 ON table1.column_name=table2.column_name;	FOREACH (JOIN alias1 BY column_name LEFT|RIGHT|FULL, alias2 BY column_name) GENERATE column_name(s);
UNION ALL	SELECT column_name(s) FROM table1 UNION ALL SELECT column_name(s) FROM table2;	UNION alias1, alias2;
AVG	SELECT AVG(column_name) FROM table_name;	FOREACH (GROUP alias ALL) GENERATE AVG(alias.column_name);
COUNT	SELECT COUNT(column_name) FROM table_name;	FOREACH (GROUP alias ALL) GENERATE COUNT(alias);

COUNT DISTINCT	SELECT COUNT(DISTINCT column_name) FROM table_name;	FOREACH alias {
		unique_column = DISTINCT column_name;
		GENERATE COUNT(unique_column);
		};

MAX	SELECT MAX(column_name) FROM table_name;	FOREACH (GROUP alias ALL) GENERATE MAX(alias.column_name);
MIN	SELECT MIN(column_name) FROM table_name;	FOREACH (GROUP alias ALL) GENERATE MIN(alias.column_name);
SUM	SELECT SUM(column_name) FROM table_name;	FOREACH (GROUP alias ALL) GENERATE SUM(alias.column_name);
HAVING	... HAVING aggregate_function(column_name) operator value;	FILTER alias BY aggregate_function(column_name) operator value;
UCASE/UPPER	SELECT UCASE(column_name) FROM table_name;	FOREACH alias GENERATE UPPER(column_name);
LCASE/LOWER	SELECT LCASE(column_name) FROM table_name;	FOREACH alias GENERATE LOWER(column_name);
SUBSTRING	SELECT SUBSTRING(column_name,start,length) AS some_name FROM table_name;	FOREACH alias GENERATE SUBSTRING(column_name, start, start+length) as some_name;
LEN	SELECT LEN(column_name) FROM table_name;	FOREACH alias GENERATE SIZE(column_name);
ROUND	SELECT ROUND(column_name,0) FROM table_name;	FOREACH alias GENERATE ROUND(column_name);

Figure 2 Rule Set 2 for Query Conversion

Contributions

Student Name	Student ID	Contribution
LIUNARDO Mikhael	53647374	100%
LAO Choi Hin	54045244	100%
BICHER Omar	40101040	100%
GUERIN Vincent	40101026	100%
SANJEEV Karan	53682850	100%

Individual Contribution Statement
Mikhael Liunardo (53647374)

The TripAdvisor Data Processing Project is a project which attempts to compute the large scale of dataset and analyze the semantics association of the review comments with the numerical quantification of the review.

The TripAdvisor project consists of several stages of data processing which were distributed evenly among the team member. Each team member receives equal workload and is assigned to a specific duty in the development process of the TripAdvisor Data Processing Project.

In this project, my primary task and responsibility is to develop the Java Code Map-Reduce programs which process the raw text documents from TripAdvisor review comments into structured TFIDF computed data. The responsibility scope includes performing data pre-processing including corpus cleaning; punctuation and stop words elimination, formatting and JSON data structure handling.

The developed Map Reduce code will process the text information and computes the TFIDF value for each token (word) in a document. There are 4 jobs in total to produce the final computed TFIDF dataset; word count, occurrence count, document count and TFIDF computation. Each job has a dedicated Mapper and Reducer class which main function is specific to each job's goals and objectives.

There are 3 approaches being used to perform the TFIDF Map Reduce computation; Naïve java code using HashMap data structure, Non-optimized Map Reduce and Optimized Map Reduce (directly hacking into the memory management for local combiner jobs). The 3 approaches were executed separately and its result is compiled together for comparison.

Aside from the Map Reduce program development, I also aid in the development of the Pig Queries. The other team member responsibility includes pig queries development, developing the Naïve Java code with HashMap data structure and reporting and final presentation design.

Throughout this project I have learned the "difficulty and challenges" in computing large data set. When the data size becomes large, proper management and computation is required in order for the job to run smoothly and without heavily penalizing the computer memory workload. Through the experience of utilizing Clusters of Nodes in processing the dataset, I also gain the understanding on how distributed computer functions together to compute a single job. Side learning includes further exposure to Unix Environment (screen command line is one of the new technique explored throughout the project development).

In conclusion, the TripAdvisor Project experience has been very fruitful. I was able to gain hands on experience on developing Map Reduce program codes whilst developing Pig Latin queries to extract unique information from large datasets. The work distribution among the team is done in a fair and responsible manner, each team member is supportive and performs a specific role within the project development.

CS4480 – Data Intensive Computing – Project
Individual Contribution Statement
LAO Choi Hin (54045244)

In this project, I was mainly take part in two components: parse raw JSON data for analysis and developing the first two Pig queries (Controversy and Contradiction).

For JSON parsing, it is basically a supportive work for my teammate Mikhael Liunardo, who focused on data preparation, cleanup and TFIDF analysis. To do this job, I wrote a simple Java program, with a JSON parsing library called GSON (from Google), to loop through the entire dataset and generate a tab-delimitated text file (which is Pig-friendly). Initially, I was thinking about to use Pig directly for that purpose, since the application also comes with a JSON parser. However, due to the complexity of the JSON file, and Pig requires the exact schema of the JSON structure, the library used by Pig, Jackson, always report errors during MapReduce job runtime. After a few trials, I finally gave up and chose the traditional method. Since it is a simple I/O program, it just took around 3 minutes to run on a traditional PC. But I believe that if I can take advantage of Hadoop environment, the performance could be better.

For the two queries, the most challenging part was to develop script to calculate standard deviation in controversy query. As I am not just dealing one but multiple columns of numbers and also considering the hotel ID of each record, things go unbelievably complicated and tedious if I am going to do this in just a few lines of statements. Especially in debugging, there was one intermediate step that I need to take square of the difference of each attribute with its corresponding average. I did the difference and square all at once: (x – mean) ∗ (x – mean), and the script compilation was doing fine, but errors came out during runtime saying that two rows of record are generated. As I already wrote the entire script, it is difficult for me to locate which statement threw the error. Eventually I have to dump the result backwards, statement by statement. And finally it was found to be the problem of concurrent subtraction and multiplication. After I split the operation into two steps, the jobs run perfectly. Lesson learned.

In general, the learning experience in this project is very comprehensive. Although we did not dive into the other Hadoop components like Spark and Hive, the challenges I faced in Pig and traditional MapReduce already gave me a deep understanding of what "Big Data" actually doing. Therefore, though it looks like a contradiction, the project is inclusive but also intensive. I hope that in one day, those knowledge will make a difference on my problem solving skill.

Individual Contribution Statement

BICHR EL OUDGHIRI Omar
40101040

I did this project with my classmate Vincent GUERIN, Mikhael LIUNARDO and Choi Hin LAO.

I knew that this project would take place in good conditions. Indeed, I find that our profiles are quite complementary, and we got along well. Each one of us was very motivated and wanted to bring his ideas, and this emulsion has extended throughout the project.
We separated the tasks fairly and have successfully organized ourselves.

With GUERIN, we have developed Naïve solution using Java Program. LIUNARDO and LAO have developed a Map-Reduce Java code to calculate the TF-IDF.

Our highlight was communication and we were able to manage the pressure with calm and serenity. I'm glad I could help my partner and was able to explain the technical points that I had successfully implemented.

After have finished our code, we divided task, each one of us have developed queries using Pig Latin in our result.

I'm happy of the work done by the whole group and functionalities of our program. Indeed, the work that I have brought to this project was beneficial and allowed me to practice most of the new knowledge acquired in the course of Data Computing (especially Pig Latin, a totally new language to me).
Finally, I am satisfied with the atmosphere that was in our group and the mutual support that developed.

Individual Contribution Statement

GUERIN Vincent
40101026

The project took place in good conditions as everyone did his part of the work as we have divided work between everyone. We had a plan that we followed all along. We have separate the tasks as following :

- Omar and I did naive solution using Java and Pig queries about top words for top or worst hotels .
- Mikhael and Joe did the Map reduce program in Java to calculate TF IDF and Pig queries about anomalies.

If someone had a problem, we helped each other but both groups remain independents. We have focused on getting relevant results and we have successfully done it.

Our highlight was communication and we were able to manage the pressure with calm and serenity. I'm glad I could help my partner and was able to explain the technical points that I had successfully implemented.

We have implements all the mains and some additional steps of our plan so I am proud of the results. I have gain more knowledge about Pig as I put into practice optimizations features and I have seen the speed difference by myself. I have also understood the map reduce program of my two colleagues as they explained to us how they managed to do it. Overall, this project was good to use notions that we have seen in course all this semester.

CS4480: Data Intensive Computing
Individual Contribution Statement
Karan Sanjeev (53682850)

My role in this project was to handle query translation from SQL to Pig Latin while also optimizing the queries. The translation step from SQL to Pig Latin was done by converting SQL keywords into Pig Latin equivalent. Primarily, I tried to automate this step by writing a Python script, but later realized that complicated queries cannot be converted using this method and requires human interference to carry out the translation manually. For the Pig Latin query optimization, I used the optimization rule set and repeated this process in an iterative manner continuously until we recorded a good performance.

This project has played a pivotal role in shaping my future career as I have learnt a lot from this project while broadening my perspective on Hadoop and distributive data computation. A couple of key aspects that I gained from this project is as follows:

1. Optimization is an iterative process which is specific to the dataset/query and applying an optimization rule set iteratively will not work in all cases.

2. To get the best performance in Pig Latin query optimization, we need to take advantage of distribution of map reduce jobs.

During the completion of this project, I realized the importance of teamwork and collaboration. Our project revealed the inner working spirit and motivation to excel within each of us. This project prepared us for the better in our future careers after graduation. Finally, I would like to thank Dr. Sarana Nutanong and his team of tutors for constantly supporting us throughout the entire learning process. On a side note, I believe that the distribution of work load among the group members was fair and that each member of the group worked equally hard in order to aid the timely completion of the project. The successful completion of this project would not have been possible without the help and dedication of all the team members.