YELP DATA INTO INSIGHTS

A Project

Presented to the faculty of the Department of Computer Science
California State University, Sacramento

Submitted in partial satisfaction of
the requirements for the degree of

MASTER OF SCIENCE

in

Computer Science

by

Aditya Kankanala

FALL
2016
YELP DATA INTO INSIGHTS

A Project

by

Aditya Kankanala

Approved by:

_______________________________, Committee Chair
Dr. Ying Jin

_______________________________, Second Reader
Gita Faroughi, Professor

_______________________________
Date
Student: Aditya Kankanala

I certify that this student has met the requirements for format contained in the University format manual, and that this project is suitable for shelving in the Library and credit is to be awarded for the project.

__________________________, Graduate Coordinator

Dr. Ying Jin

Department of Computer Science
Abstract

of

YELP DATA INTO INSIGHTS

By

Aditya Kankanala

Enterprise-level data processing is taking new steps in terms of data management and to derive practical insights from large volumes of data. To stream this big data reliably into actionable results is highly effective using Hadoop technology. Hadoop MapReduce distributed framework enables processing of large datasets simultaneously on large cluster of nodes where the dataset is broken into chunks. Amazon Elastic MapReduce (EMR) gives advantage in deploying and maintaining the Amazon Elastic Compute Cloud (EC2) instances and easily retrieves the data which is stored in cloud like Amazon Simple Storage Service (S3).

In this project, I designed and implemented the MapReduce Jobs using Hadoop MapReduce framework on popular Yelp dataset provided by Yelp Inc. The purpose of the project is to extract the hidden facts from the Yelp dataset to make the entrepreneurs understand the business growth and estimate the impact of users rating over the period of time. In addition to that, from the perspective of users it finds the peak time and busy days of the business and gives the information to make early appointments to get around long wait times. Implementation of MapReduce Jobs are carried out in Amazon EMR clusters and used Amazon cloud S3 for data storage.
This project has a web application interface built using AngularJS Single Page Architecture. The results from MapReduce Jobs are shown in a graphical manner using ChartJS. This application also allows users to filter business details, compare growth of different business categories and location advantage for a business over other locations.

_______________________, Committee Chair
Dr. Ying Jin

_______________________
Date
ACKNOWLEDGEMENTS

I would like to express my gratitude to Dr. Ying Jin for being my project advisor and for her continuous support and encouragement throughout my project. Her feedback and assistance were very important for me to successfully complete my master’s project.

I would like to thank Professor Gita Faroughi for her valuable time to review my report and giving me important feedback.

I would also like to thank the Department of Computer Science at California State University, Sacramento for providing an opportunity to pursue my Master’s Degree and guiding me all the way to become a successful student.
# TABLE OF CONTENTS

<table>
<thead>
<tr>
<th>Chapter</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Acknowledgements</td>
<td></td>
<td>vii</td>
</tr>
<tr>
<td>List of Figures</td>
<td></td>
<td>xi</td>
</tr>
<tr>
<td><strong>CHAPTER</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1. INTRODUCTION</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>1.1 Goal of the Project</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>1.2 Research</td>
<td></td>
<td>2</td>
</tr>
<tr>
<td>2. BACKGROUND</td>
<td></td>
<td>4</td>
</tr>
<tr>
<td>2.1 Data</td>
<td></td>
<td>4</td>
</tr>
<tr>
<td>2.2 Technology</td>
<td></td>
<td>7</td>
</tr>
<tr>
<td>2.2.1 Hadoop MapReduce</td>
<td></td>
<td>7</td>
</tr>
<tr>
<td>2.2.2 Amazon Elastic MapReduce</td>
<td></td>
<td>7</td>
</tr>
<tr>
<td>2.2.3 Amazon S3</td>
<td></td>
<td>8</td>
</tr>
<tr>
<td>2.2.4 AngularJS</td>
<td></td>
<td>8</td>
</tr>
<tr>
<td>3. SYSTEM DESIGN</td>
<td></td>
<td>9</td>
</tr>
<tr>
<td>3.1 Overview</td>
<td></td>
<td>9</td>
</tr>
<tr>
<td>3.2 Amazon EMR Cluster Design</td>
<td></td>
<td>10</td>
</tr>
<tr>
<td>3.3 Web Application Design</td>
<td></td>
<td>11</td>
</tr>
<tr>
<td>3.3.1 Angular Components</td>
<td></td>
<td>12</td>
</tr>
</tbody>
</table>
4. SYSTEM SETUP AND STRUCTURE .............................................................. 13

4.1 Hadoop Project Setup with STS .............................................................. 13

4.1.1 Maven Project Setup ........................................................................ 13

4.1.2 Project Object Model (POM) ............................................................. 14

4.1.3 Project Structure and Dependencies .................................................. 15

4.2 Amazon S3 Setup .................................................................................. 16

4.2.1 S3 Bucket Configuration .................................................................. 16

4.2.2 S3 Data Storage Structure ............................................................... 17

4.2.3 S3 Browser - Windows Client ............................................................ 18

4.3 Web Application Setup .......................................................................... 19

4.3.1 Web Application Project Structure .................................................. 19

4.3.2 Application Package Setup .............................................................. 20

4.3.3 Web Application Dependencies ....................................................... 21

5. IMPLEMENTATION ................................................................................. 23

5.1 Hadoop MapReduce Features Implementation ..................................... 23

5.1.1 POJO Implementation ...................................................................... 23

5.1.2 Driver, Mapper and Reducer ............................................................ 26

5.1.3 Distributed Cache: Map-Side Join .................................................... 29

5.2 Deploying MapReduce Jobs in Amazon EMR Cluster .......................... 32

5.2.1 Cluster Setup Configuration ............................................................ 32
# LIST OF FIGURES

<table>
<thead>
<tr>
<th>Figures</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Figure 1. Complete System Design</td>
<td>9</td>
</tr>
<tr>
<td>Figure 2. EMR Cluster Workflow Design</td>
<td>10</td>
</tr>
<tr>
<td>Figure 3. Web Application Design</td>
<td>11</td>
</tr>
<tr>
<td>Figure 4. Maven Project Setup</td>
<td>13</td>
</tr>
<tr>
<td>Figure 5. Project Object Model Structure</td>
<td>14</td>
</tr>
<tr>
<td>Figure 6. Project Structure and Dependencies</td>
<td>15</td>
</tr>
<tr>
<td>Figure 7. Amazon S3 Bucket Configuration</td>
<td>16</td>
</tr>
<tr>
<td>Figure 8. S3 Data Buckets Structure</td>
<td>17</td>
</tr>
<tr>
<td>Figure 9. S3 Browser Window</td>
<td>18</td>
</tr>
<tr>
<td>Figure 10. Project Structure and Dependencies</td>
<td>19</td>
</tr>
<tr>
<td>Figure 11. Package.json file</td>
<td>20</td>
</tr>
<tr>
<td>Figure 12. Node modules and Dependencies</td>
<td>21</td>
</tr>
<tr>
<td>Figure 13. Distributed Cache using Map-Side Join</td>
<td>30</td>
</tr>
<tr>
<td>Figure 14. EMR Cluster Configuration</td>
<td>33</td>
</tr>
<tr>
<td>Figure 15. Adding Step to Cluster</td>
<td>34</td>
</tr>
<tr>
<td>Figure 16. Hardware Configuration</td>
<td>34</td>
</tr>
<tr>
<td>Figure 17. Cluster List View</td>
<td>36</td>
</tr>
<tr>
<td>Figure 18. Cluster Detailed View</td>
<td>37</td>
</tr>
<tr>
<td>Figure 19. Terminated Cluster with Clone Option</td>
<td>37</td>
</tr>
</tbody>
</table>
Figure 20. Expanded Jobs View of a Cluster .................................................. 38
Figure 21. Tasks List View ........................................................................ 39
Figure 22. Cluster Log - I ........................................................................ 40
Figure 23. Cluster Log - II ........................................................................ 41
Figure 24. JSON Data Format .................................................................... 42
Figure 25. Home Page ............................................................................... 43
Figure 26. Business List View ................................................................. 44
Figure 27. Filtered Business View ............................................................ 45
Figure 28. Sorted Business View ............................................................... 45
Figure 29. Bar Chart - Season Details with Star ratings ........................... 46
Figure 30. Pie Chart - Season Details with Star ratings ............................ 47
Figure 31. Bar Chart - Season Details with Review ratings ....................... 48
Figure 32. Pie Chart - Season Details with Review ratings ....................... 49
Figure 33. Bar Chart - Popularity based on Categories .............................. 50
Figure 34. Pie Chart - Popularity based on Categories .............................. 51
Figure 35. Bar Chart - Popularity based on States .................................. 51
Figure 36. Pie Chart - Popularity based on States .................................. 52
1. INTRODUCTION

1.1 Goal of the Project

The objective of this project is to take advantage of Hadoop MapReduce framework [1] to capture the valuable insights from Yelp academic challenge dataset. The goal of Hadoop MapReduce Jobs is to make it easy for the users to find and reduce the long wait times of any business, by extracting the peak hours of businesses. Moreover, it allows Entrepreneurs to analyze different categories of business and their growth in comparison to business at multiple locations.

The MapReduce Jobs involve following subtasks and expected results:

1. Identify peak hours and busy days of the businesses.

2. Allows users to find the best time to visit a business and make an appointment at their convenience.

3. Gives business information and above two results as a single entity and lets the users to compare similar businesses easily.

4. Identifies the growth of different business categories at different locations and their success rate using review ratings.

5. Helps the new Entrepreneurs to make a better choice before investing on a particular business.
6. Identifies the existing business success rate and growth over the years using users’ ratings.

7. Shows the above results using ‘Single Page’ web application by making http requests on JSON files and above results are shown in a graphical representation using charts.

This report is structured as follows: Chapter 2 gives the detailed background information on data and technology stack that is used in the project implementation. Chapter 3 discusses about the system design details of MapReduce Jobs, EMR cluster deployment and the Angular MVC web application. Chapter 4 illustrates the complete setup and structure of MapReduce project, Amazon S3 and AngularJS MVC architecture. Chapter 5 contains the implementation of the application, that includes features execution in Spring Tool Suite, deployment of compiled jobs on an EMR cluster and workflow of web application. Chapter 6 talks about the summary and lessons learnt throughout the project experience.

1.2 Research
A great deal of research was carried out to find the right dataset, which must involve huge computation to bring out useful results. By studying the existing and ongoing projects on yelp dataset, I figured out that it has potential business information. I worked
on MapReduce programming design to extract insightful features from yelp dataset. I did simple engineering on cluster deployment and found that Amazon EMR framework with EC2 instances has less setup cost and time consumption. In addition to it, I also observed that the data storage management with Amazon S3 cloud has great performance. I adapted the above technologies to implement my project. Apart from the above, I also researched about the front-end technologies, to visualize the project results using charts. I did test the basic applications that were built using AngularJS and decided to use this framework with Single Page Architecture to implement the web application.
2. BACKGROUND

2.1 Data

‘Yelp Dataset Challenge’ [2] contains valuable local business information which is provided by Yep Inc, and it is made open source to build projects, that bind to ‘Yelp Dataset Challenge’ terms of use. The main purpose of this Yelp Dataset Challenge is to give opportunity to students who would like to research in an innovative way and find interesting results from using different approaches. This Yelp Dataset Challenge has completed six rounds and is conducting its seventh round with updated dataset as of Jan 2016. In this project, the dataset that is being used is current to seventh round of Yelp dataset challenge.

The Dataset is compiled with five JSON files. Each file has JSON-Object per line and the structure of each data file is as follows:

a) business

```json
{
    'type': 'business',
    'business_id': (encrypted business id),
    'name': (business name),
    'neighborhoods': [(hood names)],
    'full_address': (localized address),
    'city': (city),
    'state': (state),
    'latitude': latitude,
    'longitude': longitude,
    'stars': (star rating, rounded to half-stars),
}
```
'review_count': review count,
'categories': [(localized category names)],
'open': True / False (corresponds to closed, not business hours),
'hours': {
    (day_of_week): {
        'open': (HH:MM),
        'close': (HH:MM)
    },
    ...
},
'attributes': {
    (attribute_name): (attribute_value),
    ...
},
}

b) review

{
    'type': 'review',
    'business_id': (encrypted business id),
    'user_id': (encrypted user id),
    'stars': (star rating, rounded to half-stars),
    'text': (review text),
    'date': (date, formatted like '2012-03-14'),
    'votes': {(vote type): (count)},
}

c) user

{
    'type': 'user',
    'user_id': (encrypted user id),
    'name': (first name),
    'review_count': (review count),
    'average_stars': (floating point average, like 4.31),
    'votes': {(vote type): (count)},
    'friends': [(friend user_ids)],
}
'elite': [(years_elite)],
'yelping_since': (date, formatted like '2012-03'),
'compliments': {
    (compliment_type): (num_compliments_of_this_type),
    ...
},
'fans': (num_fans),
}
d) check-in

{
    'type': 'checkin',
    'business_id': (encrypted business id),
    'checkin_info': {
        '0-0': (number of checkins from 00:00 to 01:00 on all Sundays),
        '1-0': (number of checkins from 01:00 to 02:00 on all Sundays),
        ...
        '23-6': (number of checkins from 23:00 to 00:00 on all Saturdays)
    }, # if there was no checkin for a hour-day block it will not be in the dict
}
e) tip

{
    'type': 'tip',
    'text': (tip text),
    'business_id': (encrypted business id),
    'user_id': (encrypted user id),
    'date': (date, formatted like '2012-03-14'),
    'likes': (count),
}
2.2 Technology

This section gives brief introduction of all the technologies used in this project.

2.2.1 Hadoop MapReduce

Hadoop MapReduce [3] is a software framework designed for processing large datasets parallelly on large cluster of nodes. The dataset is broken into chunks of data and then processed in parallel. MapReduce at core involves two processes. Firstly, ‘Map function’ which works on relatively small amounts of data in parallel that processes the key/value pairs to generate an intermediate records of key/value pair. Second, ‘Reduce function’ accepts the intermediate records from Map function and then merges all the records with the same key. MapReduce libraries are available in most programming languages and this framework is easy to adapt. Many applications and tools have used MapReduce to provide parallel and distributed processing [1] [3] of large datasets, to users who have no experience with distributed systems.

2.2.2 Amazon Elastic MapReduce

Amazon EMR [4] allows to process large amounts of data in an effective manner with minimum steps involved. This EMR web service provided by Amazon is easy to adapt and is cost-effective. It simplifies the big data processing with managed Hadoop MapReduce framework. These data processing jobs can be easily distributed across the Amazon EC2 instances and can easily scale up these instances. EMR also allows to
configure other frameworks like Apache Spark, Presto and can quickly allow to interact with AWS data stores (like Amazon S3, Amazon Dynamo DB). Amazon EMR handles the big data complex use cases such as machine learning, web indexing and data warehousing in a secure and reliable way.

2.2.3 Amazon S3

Amazon Simple Storage Service (S3) [5] is a highly reliable and secure storage for developers and provides this service with no minimum pay and setup cost. This cloud storage interface facilitates to store and retrieve the data from anywhere on the web.

2.2.4 AngularJS

AngularJS [6] is a JavaScript framework that helps to build web applications. This framework works on dependency injection and two-way binding. It also provides rich HTML directives binds with scope variables and provides instant view changes all over the Document Object Model (DOM).
3. SYSTEM DESIGN

3.1 Overview

A complete system design is shown in Figure 1, that illustrates the workflow between the major components of the application. In simple terms, data storage is done by uploading data from local host into Amazon S3 cloud. MapReduce Jobs are shipped to Elastic MapReduce cluster as JAR files. The output from the cluster is stored back in S3. This output is rendered and shown in graphical manner using web application, which is hosted on local machine.

![Diagram of Complete System Design](image)

Figure 1. Complete System Design
3.2 Amazon EMR Cluster Design

In Amazon EMR Cluster Design shown in Figure 2, the complete data uploading from PC to Amazon S3 and its storage structure are depicted. This design also shows how to input JAR files to the EMR cluster.

![Figure 2. EMR Cluster Workflow Design](image)

As shown in Figure 2, the key steps of EMR cluster design are as follows:

- Initially, Yelp data set is acquired from Yelp Inc website and uploaded into Amazon S3 - Input Data bucket using ‘S3 browser’.
- Implement MapReduce Jobs on a Linux platform using Spring Tool Suite and export them as JAR files into JAR file bucket in S3.
- Configure EMR Cluster by specifying the data input, JAR file paths and output path as well i.e., Output bucket.
- After the completion of EMR Cluster tasks, write the results in Output bucket.
- Along with results, cluster also writes the log for the tasks that are being handled as per the configuration and stores them in Log bucket.

3.3 Web Application Design

Figure 3. Web Application Design
The AngularJS MVC architecture from Figure 3 shows the results from MapReduce(MR) Jobs that are graphically represented using angular components. Web application allows users to search the business details using filters and can perform sorting on business attributes. It gives the option to see more about the growth of a business with timeline and popularity of the business categories in different locations using pie and bar charts.

3.3.1 Angular Components

The angular components from Figure 3 are explained as follows:

- Angular Services (Ng-Services) are used to request the MR output JSON files and act as resource for rest of the components.

- Angular Controllers(Ng-Controllers) [6] can access any resource within the application and control the data from the resources.

- Angular Model maintains the binding between controllers and views and lets the views to visualize the data using controllers.
4. SYSTEM SETUP AND STRUCTURE

This section discusses the project setup and structure involved at different stages of big data development life cycle.

4.1 Hadoop Project Setup with STS

4.1.1 Maven Project Setup

To implement the Hadoop MapReduce (MR) Jobs, I’ve setup a Maven project of ‘archetype - quickstart’ in Spring Tool Suite that is shown in Figure 4 and implemented the MR jobs using Java language. Maven [7] manages the Java projects with great build automation tool that is built on top of Hadoop APIs.
4.1.2 Project Object Model (POM)

Maven based Hadoop Projects are defined using Project Object Model (POM.XML file)

```
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/maven-v4_0_0.xsd">
  <properties>
    <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
  </properties>
  <dependencies>
    <dependency>
      <groupId>junit</groupId>
      <artifactId>junit</artifactId>
      <version>3.8.1</version>
    </dependency>
    <dependency>
      <groupId>org.apache.hadoop</groupId>
      <artifactId>hadoop-common</artifactId>
      <version>2.7.6</version>
    </dependency>
    <dependency>
      <groupId>org.apache.hadoop</groupId>
      <artifactId>hadoop-mapreduce-client-core</artifactId>
      <version>2.7.1</version>
    </dependency>
    <dependency>
      <groupId>org.apache.hadoop</groupId>
      <artifactId>hadoop-mapreduce-client-common</artifactId>
      <version>2.7.1</version>
    </dependency>
    <dependency>
      <groupId>org.apache.hadoop</groupId>
      <artifactId>hadoop-yarn-common</artifactId>
      <version>2.7.1</version>
    </dependency>
    <dependency>
      <groupId>com.google.code.gson-simple</groupId>
      <artifactId>json-simple</artifactId>
      <version>1.1.1</version>
    </dependency>
  </dependencies>
</project>
```

Figure 5. Project Object Model Structure

POM [7] file of this project is shown in Figure 5, which is the basic component of the project. It defines the project information and configures the required dependencies. POM defines the dependencies using

- <groupId> - Represents the top-level package
- <artifactId> - Represents the name of the project
- <version> Represents the version of the project
4.1.3 Project Structure and Dependencies

Project Dependencies [7] defined in POM file are acquired when the build process is started and places all the dependencies under Maven Dependencies folder. Figure 6 shows the key project dependencies for Hadoop MapReduce programs.

1. hadoop-common
2. hadoop mapreduce-client-core
3. hadoop mapreduce-client-common
4. hadoop-yarn-common
5. json-simple.

Figure 6. Project Structure and Dependencies
The project structure from Figure 6 has multiple packages and each package contains MapReduce programs. This structure allows package imports and eliminates code duplication.

4.2 Amazon S3 Setup

4.2.1 S3 Bucket Configuration

Amazon S3 [5] gives a free cloud storage of 5GB for new users and configuring S3 has made easy with UI. Amazon S3 stores data in ‘buckets’ which is an object container.

Figure 7 shows the S3 bucket creation with mandatory region selection. I chose a region geographically close to me, which helped to optimize latency and reduce storage costs.
All bucket names follow lower-case naming convention and configured logging for each bucket.

4.2.2 S3 Data Storage Structure

Figure 8 illustrates the data storage structure in S3. This data structure has multiple buckets that are created to identify and maintain the separation between buckets based on their data type.

- **yelp-input** bucket - Contains the complete yelp dataset (JSON files)
- **yelp-jar-files** bucket - Contains compiled MapReduce programs as one JAR file
18

- yelp-logs - Contains logs of Cluster setup and execution
- yelp-output bucket - Contains Output of the Executed Cluster with jobs

4.2.3 S3 Browser - Windows Client

![S3 Browser Window](image)

Figure 9. S3 Browser Window

S3 browser [8] in windows client gives simple web services. Figure 9 shows the data uploading window of S3 browser which includes pending upload tasks. S3 browser stores and retrieves data from Amazon S3 cloud. It is fast and easy to upload large data files to
S3 buckets and download as well. Input data and JAR files are uploaded into S3 buckets, they are:

1. Yelp data set into “yelp-input” bucket
2. JAR files into “yelp-jar-files” bucket.

4.3 Web Application Setup

4.3.1 Web Application Project Structure

Project Structure and its dependencies of the web application are shown in Figure 10.

![Figure 10. Project Structure and Dependencies](image-url)
The web application project is built using Single Page MVC Architecture [6]. It loads only one page at a time for each request. AngularJS framework is used to build this application. Figure 10 is the AngularJS project structure which shows clear segregation of components, dependencies and libraries. Key components and their purpose in the project structure are described as follows:

- **app.js** - This JavaScript file is the root of web application
- **services** - contains data requests and loaded data objects, shared across controllers
- **controllers** - contains the action code and manipulates the views accordingly
- **views** - contains multiple html files with a controller for each view or html file

### 4.3.2 Application Package Setup

```json
{
   "name": "yelp_project",
   "description": "Yelp project front end with AngularJS",
   "version": "0.0.0",
   "homepage": "",
   "license": "MIT",
   "private": true,
   "dependencies": {
      "angular": "~1.5.0",
      "angular-route": "~1.5.0",
      "angular-loader": "~1.5.0",
      "angular-mocks": "~1.5.0",
      "html5-boilerplate": "^5.3.0"
   }
}
```

Figure 11. Package.json file
Package.json file is the project setup file which is shown in Figure 11 that includes angular, angular-route, html5-boilerplate and chartjs dependency files to build the web application.

4.3.3 Web Application Dependencies

Figure 12 shows the web application dependencies as separate modules that are required to start an AngularJS web application.

![Node modules and Dependencies](image)

Figure 12. Node modules and Dependencies

Web application dependency folders are loaded while building the application. Figure 12 represents the dependency files as follows:

- bower_components - Bower manager is used to fetch and install specified un-minified dependencies in the package.json file.
- node_modules - Node.js [9] is a module loading system, that compiles unminified dependency files and stores minified versions under node_modules folder.

- lib - Library resources represents the external dependencies which include MapReduce output JSON files and UI styling libraries
5. IMPLEMENTATION

5.1 Hadoop MapReduce Features Implementation

MapReduce Programming model was used to implement the features whose mechanism involved the following steps,

- Parsed the Yelp Data files and stored them as Plain Old Java Objects (POJOs)
- Passed the POJOs to respective Mapper and Reducer functions and implemented the logic to retrieve outputs from a big chunk of data
- Combined the outputs of two different MapReduce jobs using Distributed Cache Map side join

More details of above steps are discussed in the following sections using code snippets.

5.1.1 POJO Implementation

Parser classes are simple java classes which are used to parse the yelp data files and store each line of data as POJO. These Parser classes are grouped under one package and easily imported in other feature-implemented packages. This reduced code duplication because parsing data and passing it to Mappers is required in each feature implementation.

Code Snippet: POJO Class

```java
// POJO Class for parsing Business Details from business.json file
package com.yelp.parsers;
```
public class BusinessCheckinOutputParser {
    String businessId;
    String businessName;
    String businessState;
    String businessCategory;
    String businessAppointment;
    String busyDay;

    public void parse(String record) {
        String[] vals = record.split("\t");

        this.businessId = new String(vals[0]);
        this.businessName = new String(vals[1]);
        this.businessState = new String(vals[2]);
        this.businessCategory = new String(vals[3]);
        this.businessAppointment = new String(vals[4]);
        this.busyDay = new String(vals[5]);
    }

    public BusinessCheckinOutputParser(String record){
        parse(record);
    }

    public String getBusinessId() {
        return businessId;
    }

    public void setBusinessId(String businessId) {
        this.businessId = businessId;
    }

    public String getBusinessName() {
        return businessName;
    }

    public void setBusinessName(String businessName) {
        this.businessName = businessName;
    }

    public String getBusinessState() {

public void setBusinessState(String businessState) {
    this.businessState = businessState;
}

public String getBusinessCategory() {
    return businessCategory;
}

public void setBusinessCategory(String businessCategory) {
    this.businessCategory = businessCategory;
}

public String getBusinessAppointment() {
    return businessAppointment;
}

public void setBusinessAppointment(String businessAppointment) {
    this.businessAppointment = businessAppointment;
}

public String getBusyDay() {
    return busyDay;
}

public void setBusyDay(String busyDay) {
    this.busyDay = busyDay;
}

5.1.2 Driver, Mapper and Reducer

For each feature implementation, I used three key classes for executing MapReduce Jobs. They are:

- **Driver** - This Class contains the configuration of Mapper and Reducer Classes and their input and output type definitions. The main class in Driver program contains the ‘Job class’ instance which starts the Driver program execution.

- **Mapper** - This Class parses the data file using Driver Class arguments and POJO Class. It transforms the parsed data into intermediate records using Mapper Logic.

- **Reducer** - This Class reads the intermediate results from Mapper Class and performs the reducer logic. This Reducer class contains the key data manipulation logic and writes the output in specified path after successful execution.

Following Code Snippets illustrate the above classes:

Code Snippet: Driver Class

```java
package com.yelp.findPopularity;

import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
import org.apache.hadoop.util.Tool;
import org.apache.hadoop.util.ToolRunner;
```
public class PopularityCategoryDriver extends Configured implements Tool{

    public int run(String[] arg0) throws Exception {

        Job job = Job.getInstance(getConf());

        job.setJarByClass(PopularityCategoryDriver.class);
        job.setJobName("PopularityCategoryDriver");

        FileInputFormat.setInputPaths(job, new Path(arg0[1]));
        FileOutputFormat.setOutputPath(job, new Path(arg0[2]));

        job.setMapperClass(PopularityCategoryMapper.class);
        job.setReducerClass(PopularityCategoryReducer.class);

        job.setOutputKeyClass(Text.class);
        job.setOutputValueClass(Text.class);

        return job.waitForCompletion(true)?0:1;
    }

    public static void main(String[] args) throws Exception {
        if (args.length != 3) {
            System.err.println("Error: PopularityDriver arguments");
            System.exit(-1);
        }
        System.exit(ToolRunner.run(new PopularityCategoryDriver(), args));
    }
}
Code Snippet: Mapper Class

// Mapper Class Skeleton with map() method

```java
package com.yelp.findPopularity;
import java.io.IOException;
import java.util.ArrayList;
import java.util.List;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Mapper;
import org.json.simple.JSONObject;
import org.json.simple.parser.JSONParser;

public class PopularityCategoryMapper extends Mapper<LongWritable, Text, Text, Text> {
    public void map(LongWritable longText, Text value, Context context) throws IOException, InterruptedException {

    }
}
```

Code Snippet: Reducer Class

// Reducer Class Skeleton with reduce() method

```java
package com.yelp.findPopularity;
import java.util.HashMap;
import java.util.Map;
import java.util.Iterator;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Reducer;
import com.yelp.parsers.ReviewNStarParser;

public class PopularityCategoryReducer extends Reducer<Text, Text, Text, Text> {
    public void reduce(Text key, Iterable<Text> values, Context context) throws IOException, InterruptedException {

    }
}
```
IOException, InterruptedException {
    for (Text value : values) {
        System.out.println("Entered reducer");
    }
}

Cleanup () method in Reducer Class is called at the end of the program and writes the output to the context only once.

Code Snippet: cleanup () in Reducer Class

```java
public void cleanup(Context context) throws IOException, InterruptedException {
    Iterator<Map.Entry<String, String>> iterator = reviewNstars.entrySet().iterator();
    while (iterator.hasNext()) {
        Map.Entry<String, String> entry = iterator.next();
        reduceKey.set(entry.getKey());
        ReviewNStarParser finalObj = new ReviewNStarParser(entry.getValue().toString());
        output_reviewcount = finalObj.getReview_count();
        output_starrating = finalObj.getStar_rating();
        output_businessCount = category_businesCount.get(entry.getKey());
        combineOutput = String.valueOf(output_reviewcount) + "\t" + String.valueOf(output_starrating) + "\t" + String.valueOf(output_businessCount);
        reduceValue.set(combineOutput.toString());
        context.write(reduceKey, reduceValue);
    }
}
```

5.1.3 Distributed Cache: Map-Side Join

The Distributed Cache Utility [10] is used to cache the files like text, jar etc. on map or reduce side nodes, and perform the computation as needed by the applications.
The Distributed Cache Utility is used to integrate the business details output with business peak times and busy days’ outputs. Figure 13 explains the steps involved in this process:

- Passed the small output set i.e., business details output to distribute on map side nodes.
- Parsed the busy times and days output using mapper class and combined the current read data with business details output (local copy on each node).

**Figure 13. Distributed Cache using Map-Side Join**

Code Snippet: Map-Side Join using Mapper Only Class

```java
package com.yelp.joinAppointmentDetails;

import java.io.BufferedReader;
```
import java.io.File;
import java.io.FileInputStream;
import java.io.IOException;
import java.io.InputStreamReader;
import java.util.HashMap;
import java.util.Map;
import org.apache.hadoop.io.IOUtils;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Mapper;
import com.yelp.parsers.BusinessCheckinOutputParser;

public class JoinAppointmentDistCacheMapper extends Mapper<LongWritable, Text, Text, Text> {
    private static BusinessBusyDayParser busydayparser = null;
    private static Text mapKey = new Text();
    private static Text mapValue = new Text();
    private static BusinessCheckinOutputParser outputparser = null;
    private Map<String, BusinessCheckinOutputParser> busniessDetailsjoin = new HashMap<String, BusinessCheckinOutputParser>();

    void initialize() throws IOException {
        File companyDataFile = new File("businessBusyDay1.csv");
        BufferedReader in = null;
        try {
            in = new BufferedReader(new InputStreamReader(new FileInputStream(companyDataFile)));
            String line;
            while ((line = in.readLine()) != null) {
                outputparser = new BusinessCheckinOutputParser(line);
            }
        }
    }
System.out.println("outputparser read");
busniessDetailsjoin.put(outputparser.getBusinessId(), outputparser);
System.out.println("Hashmap added");
}
} finally {
    IOUtils.closeStream(in);
}
}

protected void setup(Context context) throws IOException, InterruptedException {
    initialize();
}

5.2 Deploying MapReduce Jobs in Amazon EMR Cluster

After the MapReduce Jobs are implemented, I exported them as JAR files into Amazon S3 yelp-jar-files bucket. The JAR file was used to configure an EMR Cluster with step execution.

5.2.1 Cluster Setup Configuration

Figure 14, 15, 16 shows cluster software and hardware configuration:
Figure 14. EMR Cluster Configuration
Important values from Figure 14, 15, 16 that are required to configure an EMR cluster are defined as follows:
• Cluster Name - Specified Cluster Name to identity the cluster jobs.

• Logging - Enabled and pointed to yelp-logs bucket

• Launch mode - Step Execution
  
  ○ Step type - Custom JAR file (yelpcomplete.jar)
  
  ○ JAR location - Pointed JAR file i.e., in yelp-jar-files bucket
  
  ○ Arguments - <main_class_name> <data_input_path> <output_path>
    
    ■ <main_class_name> - Executes specified driver class from project
    
    ■ <data_input_path> - Reads the input data from this path
    
    ■ <output_path> - Writes output to this path.

• Software Configuration Release - emr 4.4.0 (Vendor - Amazon)

• Hardware Configuration - Instance type - m3.xlarge (default)
  
  No of Instances - 3 (master node - 1, slave nodes - 2)

Launched the Clusters with above configurations for each MapReduce Driver Classes.
5.2.2 Run and Monitor the Deployed Cluster

Amazon EMR clusters

- Cluster list - List of all Clusters are shown with ID’s, Status and elapsed time.
- Each Driver Class is configured as separate cluster and deployed with Step execution.
- Cloned a deployed cluster to test the output data using “Clone” button.
- Additional Steps can be added while a cluster is in executing mode.

Figure 17 shows the cluster list view:

![Cluster List View](image)

**Figure 17. Cluster List View**
Figure 18, 19 illustrates the Cluster details and execution status with elapsed time:

Figure 18. Cluster Detailed View

Figure 19. Terminated Cluster with Clone Option
The terminated cluster shown in Figure 20 gives the detailed view of jobs included in this cluster. It also specifies the total time for each job with start time and informs the action took upon cluster execution failure. I did set, action on failure to “Terminate Cluster”.

Figure 20. Expanded Jobs View of a Cluster
Each MapReduce Job is divided into set of map tasks and reduce tasks. Every task is carried out simultaneously and their status is tracked. In Figure 21 we can see the task list view, that has 23 tasks which are processed in a MapReduce Job. All the below tasks are succeeded and output is written in yelp-output bucket.

![Figure 21. Tasks List View](image)
5.2.3 Log Analysis

Cluster log files from Figure 22, 23 gives detailed version of actions with specific times and status of map jobs and reduce jobs with percentages. In the event of failure, these logs give complete details of errors. In addition to these it also shows the cluster execution status with important details like CPU time, number of bytes read and number of bytes written etc.

2016-09-15 00:27:27,055 INFO com.amazon.ws.eel.hadoop.fs.EmFileSystem (main): Consistency disabled,
2016-09-15 00:27:27,067 INFO amazon.eel.metrics.MetricsSaver (main): MetricConfigRecord disabledInC
2016-09-15 00:27:27,860 INFO amazon.eel.metrics.MetricsSaver (main): Created MetricsSaver j=3995101
2016-09-15 00:27:30,763 INFO com.amazon.ws.eel.hadoop.fs.EmFileSystem (main): Consistency disabled,
2016-09-15 00:27:30,417 INFO org.apache.hadoop.mapreduce.lib.input.FileInputFormat (main): Total inp
2016-09-15 00:27:42,328 INFO org.apache.hadoop.mapreduce.Job (main): The url to track the job: http://
2016-09-15 00:28:12,288 INFO org.apache.hadoop.mapreduce.Job (main): map 6% reduce 0%
2016-09-15 00:28:47,707 INFO org.apache.hadoop.mapreduce.Job (main): map 1% reduce 0%
2016-09-15 00:28:50,743 INFO org.apache.hadoop.mapreduce.Job (main): map 2% reduce 0%
2016-09-15 00:28:53,778 INFO org.apache.hadoop.mapreduce.Job (main): map 3% reduce 0%
2016-09-15 00:28:54,800 INFO org.apache.hadoop.mapreduce.Job (main): map 5% reduce 0%
2016-09-15 00:29:34,549 INFO org.apache.hadoop.mapreduce.Job (main): map 6% reduce 0%
2016-09-15 00:29:37,877 INFO org.apache.hadoop.mapreduce.Job (main): map 7% reduce 0%
2016-09-15 00:29:39,592 INFO org.apache.hadoop.mapreduce.Job (main): map 9% reduce 0%
2016-09-15 00:30:05,702 INFO org.apache.hadoop.mapreduce.Job (main): map 11% reduce 0%
2016-09-15 00:30:08,806 INFO org.apache.hadoop.mapreduce.Job (main): map 12% reduce 0%
2016-09-15 00:30:09,814 INFO org.apache.hadoop.mapreduce.Job (main): map 14% reduce 0%
2016-09-15 00:30:26,012 INFO org.apache.hadoop.mapreduce.Job (main): map 15% reduce 0%
2016-09-15 00:30:42,058 INFO org.apache.hadoop.mapreduce.Job (main): map 16% reduce 0%
2016-09-15 00:30:47,055 INFO org.apache.hadoop.mapreduce.Job (main): map 18% reduce 0%
2016-09-15 00:31:16,245 INFO org.apache.hadoop.mapreduce.Job (main): map 19% reduce 0%
2016-09-15 00:31:13,271 INFO org.apache.hadoop.mapreduce.Job (main): map 20% reduce 0%
2016-09-15 00:31:18,204 INFO org.apache.hadoop.mapreduce.Job (main): map 23% reduce 0%
2016-09-15 00:31:42,470 INFO org.apache.hadoop.mapreduce.Job (main): map 24% reduce 0%
2016-09-15 00:31:44,508 INFO org.apache.hadoop.mapreduce.Job (main): map 25% reduce 0%
2016-09-15 00:31:49,526 INFO org.apache.hadoop.mapreduce.Job (main): map 20% reduce 0%
2016-09-15 00:31:51,546 INFO org.apache.hadoop.mapreduce.Job (main): map 27% reduce 0%
2016-09-15 00:31:59,737 INFO org.apache.hadoop.mapreduce.Job (main): map 30% reduce 0%

Figure 22. Cluster Log - I
Figure 23. Cluster Log - II
5.3 Web Application Implementation

This project implemented a web application using the MVC Architecture in combination with AngularJS [6] and ChartJS [11].

5.3.1 JSON Data Format

In order to represent the data as graphs, the output data that is retrieved from “yelp-output” bucket was formatted. Default output data was in text format and it is transformed into JSON objects, which is a preferred data format in AngularJS applications. Figure 24 shows a block of JSON object used in the application:

```
[{
  "bID": "--1emgg8hgo6ipd_RWb-g",
  "bName": "Sinclair",
  "bState": "NV",
  "bCategory": "["Food"],"Convenience Stores"]",
  "bAppointment": "false",
  "bDay": "Thursday",
  "bTime": "3 pm to 4 pm"
},
{
  "bID": "--4pe8B26g37FL5mU8g",
  "bName": "Office Max",
  "bState": "AZ",
  "bCategory": "["Shopping"],"Office Equipment"]",
  "bAppointment": "false",
  "bDay": "Tuesday",
  "bTime": "5 pm to 6 pm"
},
{
  "bID": "--5j83Z3-nUPZxUvrb38Uw",
  "bName": "Mika's Greek",
  "bState": "AZ",
  "bCategory": "["Greek"],"Mediterranean","Restaurants"]",
  "bAppointment": "false",
  "bDay": "Tuesday",
  "bTime": "11 am to 12 noon"
}],
```

Figure 24. JSON Data Format
Web application is segmented into three sections,

- Listed all business with their details
- Compared the location advantage for a business in relation with business categories and vice versa.
- Illustrated business growth over a period with respect to users’ ratings.

Figure 25 shows home page of web application with project background and dataset details:

![Figure 25. Home Page](image-url)
5.3.2 Businesses List View

Businesses list view represents the output data from MapReduce jobs in JSON format and it is retrieved by Angular services. Users can see about 45,000 businesses and their details. A search option is provided to search and sort the results by typing keywords in ‘search’ box and choose sort type from ‘sort by’ dropdown. In addition to basic business details, ‘View More’ button gives more information about business growth as well. The view more details are explained below in Season finder section. Figure 26, 27, 28 shows businesses list, search and sort views:

![Business List View](image)

**Figure 26. Business List View**
### Business Details

**Dataset Configuration**

Dataset size: 40166

**Search:**

- Enter Keyword

**Sort:**

- Sort By: 

Total execution time (searching and sorting): 0.106

### Business Details

**Dataset Configuration**

Dataset size: 40166

**Search:**

- Enter Keyword

**Sort:**

- Sort By: 

Total execution time (searching and sorting): 0.106

<table>
<thead>
<tr>
<th>Business ID</th>
<th>Business Name</th>
<th>State</th>
<th>Reservation</th>
</tr>
</thead>
<tbody>
<tr>
<td>TW3h3F_Mg6bEWEbEuj6Eq</td>
<td>TODO'S Sandwiches</td>
<td>AZ</td>
<td>false</td>
</tr>
<tr>
<td>nd7n47n52jcu7L6oyg68g</td>
<td>TODO'S Sandwiches</td>
<td>AZ</td>
<td>false</td>
</tr>
</tbody>
</table>

**Figure 27. Filtered Business View**

<table>
<thead>
<tr>
<th>Business ID</th>
<th>Business Name</th>
<th>State</th>
</tr>
</thead>
<tbody>
<tr>
<td>kxWvSth4h4v2s2d4d8PKw</td>
<td>#1 Brothers Pizza</td>
<td>AZ</td>
</tr>
<tr>
<td>pZUk1j4ZNo06ce890c5G</td>
<td>#1 Hawaiian Barbecue</td>
<td>NV</td>
</tr>
<tr>
<td>Mx3vT2Rt_CFEw-vCQqaEQ</td>
<td>#1 Hawaiian Barbecue</td>
<td>NV</td>
</tr>
<tr>
<td>K2KepT111Df9w9y4Q0B0g</td>
<td>#1 Sushi</td>
<td>AZ</td>
</tr>
</tbody>
</table>

**Figure 28. Sorted Business View**
5.3.3 Season Finder

When clicking ‘View More’ button in business list view on a business row, it gives options to see the growth of the business with respect to users’ star and review rating. This selection is made by selecting from the ‘Sort By’ dropdown. Figure 29, 30 shows the growth of the business star rating over a period with high’s and low’s. Moreover, displayed both bar and pie charts for each business with data marking on X and Y axes.

![Business Season Details](image)

Figure 29. Bar Chart - Season Details with Star ratings
Figure 30. Pie Chart - Season Details with Star ratings
To switch from star ratings graph view to review ratings graph view, select the appropriate option from the drop down. Now click on “Show Pie Chart” or “Show Bar Chart” button to view respective graphs. Figure 31, 32 shows the bar and pie charts of season details with review ratings.

![Business Season Details](image)

**Figure 31. Bar Chart - Season Details with Review ratings**
Business Season Details

Figure 32. Pie Chart - Season Details with Review ratings
5.3.4 Popularity Checker

In Popularity Checker, users can enter a state name and select one of the options (Business Count, Star rating, Review ratings) to view and compare the business categories growth. Users can also select category and an option to compare different states growth and success on a business category. Figure 33, 34, 35, 36 illustrates the given explanation as pie and bar charts:

Figure 33. Bar Chart - Popularity based on Categories
Figure 34. Pie Chart - Popularity based on Categories

Figure 35. Bar Chart - Popularity based on States
The above implemented web application gives complete details of the MapReduce Jobs output data in the form of graphs. This application was deployed on localhost using port 8000 to serve up the data to the angular app.
6. CONCLUSION

The purpose of the project is to retrieve interesting facts out of the Yelp dataset. It helps users to get around the long wait times at a business and to grow their business. This was achieved in the process of learning and implementing Hadoop MapReduce jobs on an Amazon EMR cluster. A web application was also developed to show the results to end users which allows users to analyze and make choices based on their need. This web application also gives the option to compare different businesses based on users’ ratings and reviews.

6.1 Lessons Learned

As part of this project, I learned a lot of concepts related to big data technologies. In the process of project implementation, I got the opportunity to learn about data processing jobs using Hadoop MapReduce framework. I gained knowledge about Amazon EMR cluster deployment and Amazon S3 Data management with best practices, as part of MapReduce jobs execution. In the process of web application implementation, I learned AngularJS application development using Single Page Architecture. On a whole I understood the big data development life cycle and web application development. This project gave an opportunity to implement the above concepts in practical, which gave me an in-depth knowledge of how to retrieve useful information from large unused datasets.
In terms of technical skills, I learned advanced Java programming skills by following code reusability and loose coupling principles. I also learnt AngularJS, a popular JavaScript framework and its design patterns to quickly build complex web applications with two-way binding.
BIBLIOGRAPHY


