

# Amazon Movie Review Analysis

## Team members:

Dev Pranav Puchakayala  
Nasir Ahmed Raffick  
Siddarthan, Visvanathan  
Pavitra Srinivasan

## Introduction

Our project focuses on analyzing Amazon Movie Review Dataset. We have successfully implemented all the tasks mentioned in the project proposal. Our first task was to create a web-based application and use the Amazon's AWS SDK-toolkit to dynamically launch clusters on AWS and read the result files from S3. So the best part of the project was that we could create clusters on button click from the front end web application and avoid going to AWS EMR in order to make the settings each time to create the clusters. The second task was to find the top k useful reviewers based on the helpfulness rating. The third task was to enhance the number of features by combining the results returned from Amazon Product Advertising API and OMDb API. The combined data was then loaded into MySQL database by using SparkSQL by filtering out the required data. The fourth task was to fetch the data from MySQL database based on user input. The records can be filtered based on IMDb rating, Amazon Review Score or number of Oscars and Awards won. The highlight of the project was the use of MySQL database instead of using SparkSQL for data retrieval. The reason behind why we went ahead with saving the data joined and filtered by SparkSQL into MySQL database because SparkSQL was non-persistent and it requires the entire computation to be done again and again for each query and hence we could not retrieve the data quickly while querying. It was possible to execute queries based on user selection through SparkSQL by creating a cluster for each query on AWS. However, since we had developed a user interface, we could not afford to make the user wait for a long time to get the results based on his/her selection on the front-end. Therefore, we went ahead with a persistent data storage using MySQL, which was very responsive from the front end in order to improve the user experience.

## Data

We used the Amazon Movie Review dataset. We obtained the dataset from the following link: <https://snap.stanford.edu/data/web-Movies.html>

This dataset consists of movie reviews from Amazon. The data span a period of more than 10 years, including all ~8 million reviews up to October 2012. The reviews include product and user information, ratings, and a plaintext review. Following statistics have been provided for this dataset.

Dataset statistics	
Number of reviews	7,911,684
Number of users	889,176
Number of products	253,059
Users with > 50 reviews	16,341
Median no. of words per review	101
Timespan	Aug 1997 - Oct 2012

File	Description	Size
movies.txt.gz	Amazon movie data (~8 million reviews)	9.3GB

### Dataset Format:

The Amazon movie review dataset is provided in movies.txt.gz file in the following text format:

```
product/productId: B00006HAXW
review/userId: A1RSDE90N6RSZF
review/profileName: Joseph M. Kotow
review/helpfulness: 9/9
review/score: 5.0
review/time: 1042502400
review/summary: Pittsburgh - Home of the OLDIES
review/text: I have all of the doo wop DVD's and this one is as good or better
than the 1st ones.
```

where

product/productId: id of the product which can be accessed as  
amazon.com/dp/B00006HAXW

review/userId: id of the user

review/profileName: name of the user

review/helpfulness: fraction of users who found the review helpful

review/score: rating of the product

review/time: time of the review (unix time)

review/summary: review summary

review/text: text of the review

The above data was converted from text format to JSON in order to be able to pass the entire details for one product to a mapper, which is not possible, if the data was kept in text format. So the JSON file was loaded into the S3 bucket.

Taking the advantage of parallelism in MapReduce, we queried the Amazon Product Advertising API using the product id information from the JSON file to get the title information for the corresponding product id from the existing dataset. The data returned by the Amazon API was in the following format:

```
<Item>
  <ASIN>B000A2XB9U</ASIN>
  <ItemAttributes>
    <Director> James Cameron </Director>
    <EAN>0014381273229</EAN>
    <Format>Color</Format>
    <Language>
      <Name>English</Name>
      <Type>Original Language</Type>
    </Language>
    <ListPrice>
      <CurrencyCode>USD</CurrencyCode>
      <FormattedPrice>$19.99</FormattedPrice>
    </ListPrice>
    <NumberOfItems>1</NumberOfItems>
    <ProductGroup>DVD</ProductGroup>
```

```

    <ReleaseDate>2009-12-18-</ReleaseDate>
    <Studio>Image Entertainment</Studio>
    <Title>Avatar</Title>
  </ItemAttributes>
</Item>

```

Using the same advantage of parallel processing, based on the Title field obtained by parsing the above XML, the OMDb API was hit to get the information about the movie/video like the Genre, Awards, IMDB rating, Language, Year etc. These fields are necessary for the third task in the web application, which was fetching the records based on user input. The data returned by the OMDb API was in the following format:

```

{
  "Title": "Avatar",
  "Year": "2009",
  "Rated": "PG-13",
  "Released": "18 Dec 2009",
  "Runtime": "162 min",
  "Genre": "Action, Adventure, Fantasy",
  "Director": "James Cameron",
  "Writer": "James Cameron",
  "Actors": "Sam Worthington, Zoe Saldana, Sigourney Weaver, Stephen Lang",
  "Plot": "When his brother is killed in a robbery, paraplegic Marine Jake Sully...",
  "Language": "English, Spanish",
  "Country": "USA, UK",
  "Awards": "Won 3 Oscars. Another 84 wins & 106 nominations.",
  "Poster": "http://ia.mediaimdb.com/images/M/M15BM15BanBnXkFTcwODc5MTUwMw._V1_SX300.jpg",
  "Metascore": "83",
  "imdbRating": "7.9",
  "imdbVotes": "818,467",
  "imdbID": "tt0499549",
  "Type": "movie",
  "Response": "True"
}

```

So the Amazon API resultant dataset was combined with the OMDDB API dataset, in order to increase the number of features in the existing dataset. And the combined dataset was used for further tasks.

## Technical discussion

### **TASK 1:** Creating a Web Interface and creating cluster using the AWS CREATE CLUSTER SDK

The purpose of this task was to create a single user web interface, through which a user can select the required tasks to be run, which are specified below. The main advantage of this task is that the cluster is created programmatically, thus allowing us to dynamically use the user's input to run the corresponding Map Reduce Task.

The link for AWS SDK to create cluster:

<http://docs.aws.amazon.com/ElasticMapReduce/latest/DeveloperGuide/callin-g-emr-with-java-sdk.html>

### **Pseudo Code:**

#### Code for launching the cluster on EMR

```
AmazonElasticMapReduceClient emr = new
AmazonElasticMapReduceClient(credentials);

StepFactory stepFactory = new StepFactory();

String[] params = {"s3://mr-project/Dataset", "s3://mr-project/Output-
task1"};

HadoopJarStepConfig reviews = new HadoopJarStepConfig()
    .withJar("s3://mr-project/task1.jar")
    .withArgs(params);

StepConfig enableddebugging = new StepConfig()
    .withName("Enable debugging")
```

```
.withActionOnFailure("TERMINATE_JOB_FLOW")
    .withHadoopJarStep(stepFactory.newEnableDebuggingStep());

StepConfig customJar = new StepConfig()
    .withName("Top k Helpful reviewers")
    .withActionOnFailure("CONTINUE")
    .withHadoopJarStep(reviews);

RunJobFlowRequest request = new RunJobFlowRequest()
    .withReleaseLabel("emr-4.0.0")
    .withName("Task 1")
    .withSteps(enableddebugging, customJar)
    .withLogUri("s3://log-project/Task1/")
    .withServiceRole("EMR_DefaultRole")
    .withJobFlowRole("EMR_EC2_DefaultRole")
    .withInstances(new JobFlowInstancesConfig()
    .withInstanceCount(11)
    .withTerminationProtected(false)
    .withKeepJobFlowAliveWhenNoSteps(false)
    .withMasterInstanceType("m1.medium")
    .withSlaveInstanceType("m1.medium"));

RunJobFlowResult result = emr.runJobFlow(request);
```

### Code for reading the files from S3

```
AmazonS3 s3client = new AmazonS3Client(credentials);

ListObjectsRequest listObjectsRequest = new ListObjectsRequest()
    .withBucketName("mr-project")
    .withPrefix(prefix);
ObjectListing objectListing;

do {
    objectListing = s3client.listObjects(listObjectsRequest);
    for (S3ObjectSummary objectSummary :
        objectListing.getObjectSummaries()) {
        if(objectSummary.getKey().contains("part-r")){

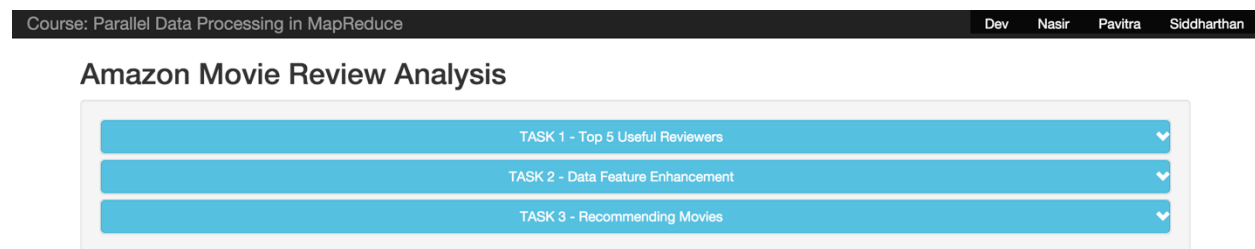
            S3Contents.getContents(objectSummary.getKey(),reviewerList,s3client);
        }
    }
}
```

```
    }  
    listObjectsRequest.setMarker(objectListing.getNextMarker());  
} while (objectListing.isTruncated());
```

### Files for this task are:

LaunchCluster.java  
frontUI.html  
result.jsp  
DisplayOptionResult.jsp  
RecieveRequest.java  
Options.java

The screenshot for task creation using Amazon Cluster SDK are as follow:



### TASK 2: Top k useful reviewers

The purpose of the task is to find the top useful reviewers whose reviews helped people buy the product, which in our case is a movie/video.

As we know from the dataset that there are 16,341 users who had done over 50 reviews are present in the dataset. This implies that a particular person would have reviewed multiple movies/videos. Moreover, multiple users would have provided the rating to the reviewer stating if it was helpful for them to buy the movie/video. Hence, we are planning to look for the top reviewers and check for their helpfulness rating. The review helpfulness field is key element for this task. So based on the average helpfulness rating, we will determine the top k

reviewers whose reviews were considered as helpful by other users based on the rating provided by the users to the reviewer. The dataset does not provide the category/genre information so we will be providing the useful reviewers overall. Therefore, we will aggregate those results to find the top k reviewers whose reviews have been helpful for motivating other users to buy that movie/video.

For the mapper, the reviewer id is the key. The profileName and the helpfulness is the value. At the reducer, the helpfulness per reviewer is calculated.

The helpfulness score is computed using the formula,

$$\log\left(\frac{1 + votes_{+ve}}{total_{movies}}\right) / \sum \frac{votes_{+ve}}{votes_{total}}$$

Where,

$votes_{+ve}$  = determines the number of positive votes a reviewer has got out of the total number of votes for all the movies that he has reviewed.

$votes_{total}$  = determines the total number of votes received by a reviewer for all the movies that he has reviewed.

$total_{movies}$  = determines the total number of movies in the given dataset

The reason why we came up with this formula is to give more weightage to the person who has majority positive reviews across the number of movies he had reviewed and considering this measure against the total number of movies in the dataset.

For e.g.: If a person has got 1 positive vote out of 1 vote that he received then his helpfulness percent would be 100% whereas a person who has got 500 positive votes out of 500 votes that he received, his helpfulness percent would also be 100%. So using the above formula, we are able to assign more positive



weight to the person who received 500 votes against the person who just reviewed 1 vote.

Also, a person who had received 90 positive votes out of 100 and another person who had received 900 positive votes out of 1000 might have the same 90% score but the person who had received 900 positive votes should have higher weight. This factor is also considered in the formula.

Also who has got 50 positive votes out of 100 for 1 movie and 50 positive votes out of 100 for 40 movies should have different weight. The second person should be given more weightage compared to first person. This factor was also considered when designing the formula.

## Pseudo Code:

Map:

```
public void map(Object key, Text value, Context context) throws Exception{
    userID = new Text(row.getString("review/userid").trim());
    helpfulness = row.getString("review/helpfulness").trim();
    profileName = row.getString("review/profilename").trim();
    Text val = new Text(helpfulness+"#"+profileName);
    context.write(userID, val);
}
```

Reduce:

```
public void reduce(Text key, Iterable<Text> values, Context context) throws
Exception{
    for(Text value : values){
        numOfReviews++;
        val = value.toString().split("#");
        String[] helpfulnessValue = val[0].split("/");
        int numerator = Integer.valueOf(helpfulnessValue[0].trim());
        int denominator = Integer.valueOf(helpfulnessValue[1].trim());
        totalNumerator += numerator;
        totalDenominator += denominator;
        profileName = val[1].trim();
    }
    if(numOfReviews >= 50 && totalNumerator != 0 && totalDenominator != 0){
        double helpfulPercent = totalNumerator/(double)totalDenominator;
        double score =
Math.log((1+totalNumerator)/(double)253059)/helpfulPercent;
        if(!Double.isNaN(score)){
            String value =
profileName+"\t"+score+"\t"+numOfReviews+"\t"+helpfulPercent+"\t"+totalNumerator+"/"+
totalDenominator;
            context.write(key, new Text(value));
        }
    }
}
```

}

**Files for this task are:**

Top5List.java

ReviewerHelperClass.java

S3Contents.java

ReadS3files.java

RecieveRequest.java

**Screenshot for task execution:** You can set the number of top records to any value from 5 to 5000.

Course: Parallel Data Processing in MapReduce
Dev
Nasir
Pavitra
Siddharthan

### Amazon Movie Review Analysis

TASK 1 - Top 5 Useful Reviewers

In this task, we are planning to analyze the "review/helpfulness" field from the dataset:

```
product/productId: B00006HAXW
review/userId: A1R5DE90N6RSZF
review/profileName: Joseph M. Kotow
review/helpfulness: 9/9
review/score: 5.0
review/time: 1042502400
review/summary: Pittsburgh - Home of the OLDIES
review/text: I have all of the doo wop DVD's and this one is as good or better than the 1st ones.
```

The dataset contains data with each user having more than 50 reviews. This implies that a particular person would have reviewed multiple movies/videos. And multiple users would have provided the rating to the reviewer stating if it was helpful for them to buy the movie/video. Hence we look for the top reviewers and check for their helpfulness rating. The review helpfulness rating is the key for this task. So based on the average helpfulness rating, we will determine the top 5 reviewers whose reviews were considered as helpful by other users based on the rating provided by the users to the reviewer. The dataset does not provide the category/genre information so we will be providing the useful reviewers overall. So we will aggregate those results to find the top 5 reviewers whose reviews have been helpful for motivating other users to buy that movie/video.

Launch Task
Number of top records: 500

TASK 2 - Data Feature Enhancement

TASK 3 - Recommending Movies

The output of clicking Launch Task is the following:



Rank	Reviewer Name	Review Counts	Review Helpfulness
1	lawrance m. bernabo	9713	139555/159440
2	grady harp	7448	113130/127719
3	gary f. taylor "gft"	3419	101455/112464
4	wayne klein "if at first the idea is not absu...	5172	99988/110440
5	lawyeraau	5745	80887/89233

Machine setup through aws sdk:

```

RunJobFlowRequest request = new RunJobFlowRequest()
    .withReleaseLabel("emr-4.0.0")
    .withName("Task 1")
    .withSteps(enableddebugging,customJar)
    .withLogUri("s3://log-project/Task1/")
    .withServiceRole("EMR_DefaultRole")
    .withJobFlowRole("EMR_EC2_DefaultRole")
    .withInstances(new JobFlowInstancesConfig()
    .withInstanceCount(11)
    .withTerminationProtected(true)
    .withKeepJobFlowAliveWhenNoSteps(false)
    .withMasterInstanceType("m1.medium")
    .withSlaveInstanceType("m1.medium"));

```

**Performance Comparison:**

11 machines (1 master 10 workers)	m1.medium	352 secs
6 machines (1 master 5workers)	m1.medium	710secs

**TASK 3: Data Feature Enhancement**

In this task, we took the product id from the original dataset and used the advantage of parallel processing to hit the Amazon Product Advertising API to get the Title for the product. During this task, the product id was the key and the review score was the value for the mapper. In the reducer, the product id and list of review scores was sent and the average review score was calculated. This was written as a file on S3. Another separate task runs for each title returned by the Amazon API, canonicalization is performed (because the title were different having type like [VHS], [DVD], [COLLECTOR EDITION] appended to the title) and the canonicalized title was sent to OMDB API to find all matching records from Amazon API, thus enhancing the features of movie records, which eventually increased the dataset size.

After getting the results from OMDB API, we got the details of the awards for each movie. The awards had variations like Won 3 Oscars, Another 4 wins and 21 nominations or Nominated for 1 Golden Gate award, and 5 nominations. There were combinations containing “wons”, “wins”, “nominations” and “nominated”. Therefore, we parsed the awards to find the individual number for wons and wins. We combined the score for nominated and nominations

together.

We used Apache SparkSQL to perform equi-join on the Amazon and OMDb API results on the title field. In addition, we filtered out the records to contain only the following fields,

Title, Genre, Amazon Average Review Score, Awards, Poster, IMDB Rating, Language, Won, Wins, Nominations.

### Pseudo Code:

#### Fetching Amazon Title:

##### Map:

```
public static class Map extends Mapper<LongWritable, Text, Text,
DoubleWritable> {
    public void map(LongWritable key, Text value, Context context) throws
    IOException, InterruptedException {
        try {
            JSONObject row = new JSONObject(value.toString());
            Text prodID;
            try{
                prodID = new Text(row.getString("product/productid").trim());
            }catch(JSONException e){
                prodID = new Text("EMPTY");
            }
            double score = 0.0;
            try{
                score = row.getDouble("review/score");
            }catch(JSONException e){
                score = 0.0;
            }
            context.write(prodID, new DoubleWritable(score));
        }
    }
}
```

##### Reduce:

```
public void reduce(Text key, Iterable<DoubleWritable> values, Context
context) throws IOException, InterruptedException {
    double avgScore=0;
    ProductSearch p = new ProductSearch();
    Node title=null;
    int i = 0;
    for(DoubleWritable avg : values){
        i++;
    }
}
```

```

        avgscore+=avg.get();
    }
    avgscore = avgscore/i;

    title = p.getMovieDetails(key.toString());
    new Text(title.getTextContent()+"\t"+avgscore));

```

## Fetching OMDb data:

### Map: (Canonicalizing title)

```

public void map(LongWritable key, Text value, Context context) throws
IOException, InterruptedException {

```

```

    String[] id_title = value.toString().split("\t");
    try{
        String line = id_title[1];
        line = line.replaceAll("\\<.*?>", "");
        line = line.replaceAll("\\(..*?\\) ?", "").trim();
        line = line.replaceAll("\\[.*?\\] ?", "").trim();
        line = line.replaceAll("^\\//\\-\\+\\.\\.\\.\\,\\*]*", "").trim();
        context.write(new Text(line), NullWritable.get());
    }
}

```

### Reduce: (Fetching response from OMDb)

```

public void reduce(Text key, Iterable<Text> values, Context context)
throws IOException, InterruptedException {
    String title = key.toString().trim();
    String amazonTitle = title.replace("\\", "").trim();
    title = title.replaceAll("^\\-\\+\\.\\.\\.\\,]", " ").trim();
    title = title.replace("\\", "");
    title = title.replaceAll(" ", "+");
    JSONParser jsonParser = new JSONParser();
    JSONObject jsonObject=null;
    for(String l : title.split("/",2)){
        String r = getResponse(l.trim());
        context.write(new Text(jsonObject.toJSONString()), NullWritable.get());
    }
}

```

Now we have two files one with Amazon title and Amazon Average Score and another file with OMDb response, we perform join operation using spark SQL on these two files.

NOTE: The crawling code mentioned above even though takes advantage of parallelism provided by Map Reduce Framework, The connection to Amazon web service and OMDb service has a limit on the requests, hence we were forced to use `thread.sleep()` to respect the website rule.

NOTE2: Spark-sql does not produce syslog files. To read the log, we have to access the master node through ssh connection. Time taken for spark-sql join from controller log: 97secs 6 m1.large machines.

### Files for this task are:

RecieveRequest.java

AwardScore.java

LaunchClusterSpark.java

Options.java

FileToDB.java

FormResultFile.java

TitleCrawl.java

Omdbcrawl.java

### Screenshot for task execution:

Course: Parallel Data Processing in MapReduce
Dev
Nasir
Pavitra
Siddharthan

## Amazon Movie Review Analysis

TASK 1 - Top 5 Useful Reviewers

TASK 2 - Data Feature Enhancement

In this task, we are going to use the "product/productId" field to first identify the product details by passing the ID to the Amazon Product Advertising API. So the product details would be given in the following format. Then by passing the "Title" and "ReleaseDate" field to the OMDb API we can get the information related to the movie. The data returned is in the following format:

<pre>&lt;Item&gt; &lt;ASIN&gt;B000AZXB9U&lt;/ASIN&gt; &lt;ItemAttributes&gt; &lt;Director&gt; James Cameron &lt;/Director&gt; &lt;EAN&gt;0014381273229&lt;/EAN&gt; &lt;Format&gt;Color&lt;/Format&gt; &lt;Language&gt; &lt;Name&gt;English&lt;/Name&gt; &lt;Type&gt;Original Language&lt;/Type&gt; &lt;/Language&gt; &lt;ListPrice&gt; &lt;CurrencyCode&gt;USD&lt;/CurrencyCode&gt; &lt;FormattedPrice&gt;\$19.99&lt;/FormattedPrice&gt; &lt;/ListPrice&gt; &lt;NumberOfItems&gt;1&lt;/NumberOfItems&gt; &lt;ProductGroup&gt;DVD&lt;/ProductGroup&gt; &lt;ReleaseDate&gt;2009-12-18&lt;/ReleaseDate&gt; &lt;Studio&gt;Image Entertainment&lt;/Studio&gt; &lt;Title&gt;Avatar&lt;/Title&gt; &lt;/ItemAttributes&gt; &lt;/Item&gt;</pre>	<pre>{   "Title": "Avatar",   "Year": "2009",   "Rated": "PG-13",   "Released": "18 Dec 2009",   "Runtime": "162 min",   "Genre": "Action, Adventure, Fantasy",   "Director": "James Cameron",   "Writer": "James Cameron",   "Actors": "Sam Worthington, Zoe Saldana, Sigourney Weaver, Stephen Lang",   "Plot": "When his brother is killed in a robbery, paraplegic Marine Jake Sully...",   "Language": "English, Spanish",   "Country": "USA, UK",   "Awards": "Won 3 Oscars. Another 84 wins &amp; 106 nominations.",   "Poster": "http://ia.media-imdb.com/images/M/M15BM15BanBnXkFtqw00c5MTUwMw._V1_SX300.jpg",   "Metascore": "83",   "imdbRating": "7.9",   "imdbVotes": "818,467",   "imdbID": "tt0499549",   "Type": "movie",   "Response": "True" }</pre>
----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------	------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

Launch Task

TASK 3 - Recommending Movies

## Step 1: Fetching the Joined Results from S3 bucket

Course: Parallel Data Processing in MapReduce

Dev Nasir Pavitra Siddharthan

## Amazon Movie Review Analysis

TASK 1 - Top 5 Useful Reviewers

TASK 2 - Data Feature Enhancement

In this task, we are going to use the "product/productId" field to first identify the product details by passing the ID to the Amazon Product Advertising API. So the product details would be given in the following format. Then by passing the "Title" and "ReleaseDate" field to the OMDb API we can get the information related to the movie. The data returned is in the following format:

<pre>&lt;Item&gt; &lt;ASIN&gt;B000A2XB9U&lt;/ASIN&gt; &lt;ItemAttributes&gt; &lt;Director&gt; James Cameron &lt;/Director&gt; &lt;EAN&gt;0014381273229&lt;/EAN&gt; &lt;Format&gt;Color&lt;/Format&gt; &lt;Language&gt; &lt;Name&gt;English&lt;/Name&gt; &lt;Type&gt;Original Language&lt;/Type&gt; &lt;/Language&gt; &lt;ListPrice&gt; &lt;CurrencyCode&gt;USD&lt;/CurrencyCode&gt; &lt;FormattedPrice&gt;\$19.99&lt;/FormattedPrice&gt; &lt;/ListPrice&gt; &lt;NumberOfItems&gt;1&lt;/NumberOfItems&gt; &lt;ProductGroup&gt;DVD&lt;/ProductGroup&gt; &lt;ReleaseDate&gt;2009-12-18&lt;/ReleaseDate&gt; &lt;Studio&gt;Image Entertainment&lt;/Studio&gt; &lt;Title&gt;Avatar&lt;/Title&gt; &lt;/ItemAttributes&gt; &lt;/Item&gt;</pre>	<pre>{   "Title": "Avatar",   "Year": "2009",   "Rated": "PG-13",   "Released": "18 Dec 2009",   "Runtime": "162 min",   "Genre": "Action, Adventure, Fantasy",   "Director": "James Cameron",   "Writer": "James Cameron",   "Actors": "Sam Worthington, Zoe Saldana, Sigourney Weaver, Stephen Lang",   "Plot": "When his brother is killed in a robbery, paraplegic Marine Jake Sully...",   "Language": "English, Spanish",   "Country": "USA, UK",   "Awards": "Won 3 Oscars. Another 84 wins &amp; 106 nominations.",   "Poster": "http://ia.media-imdb.com/images/M/M15BM15BanBnXkFtaw00c5MTUwMw_V1_SX300.jpg",   "Metascore": "83",   "IMDbRating": "7.9",   "IMDbVotes": "818,467",   "IMDbID": "tt0499549",   "Type": "movie",   "Response": "True" }</pre>
----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------	-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

Launch Task

Fetching Joined Results File From S3 Bucket...

TASK 3 - Recommending Movies

## Step 2: Processing awards data for each movie record

Course: Parallel Data Processing in MapReduce

Dev Nasir Pavitra Siddharthan

## Amazon Movie Review Analysis

TASK 1 - Top 5 Useful Reviewers

TASK 2 - Data Feature Enhancement

In this task, we are going to use the "product/productId" field to first identify the product details by passing the ID to the Amazon Product Advertising API. So the product details would be given in the following format. Then by passing the "Title" and "ReleaseDate" field to the OMDb API we can get the information related to the movie. The data returned is in the following format:

<pre>&lt;Item&gt; &lt;ASIN&gt;B000A2XB9U&lt;/ASIN&gt; &lt;ItemAttributes&gt; &lt;Director&gt; James Cameron &lt;/Director&gt; &lt;EAN&gt;0014381273229&lt;/EAN&gt; &lt;Format&gt;Color&lt;/Format&gt; &lt;Language&gt; &lt;Name&gt;English&lt;/Name&gt; &lt;Type&gt;Original Language&lt;/Type&gt; &lt;/Language&gt; &lt;ListPrice&gt; &lt;CurrencyCode&gt;USD&lt;/CurrencyCode&gt; &lt;FormattedPrice&gt;\$19.99&lt;/FormattedPrice&gt; &lt;/ListPrice&gt; &lt;NumberOfItems&gt;1&lt;/NumberOfItems&gt; &lt;ProductGroup&gt;DVD&lt;/ProductGroup&gt; &lt;ReleaseDate&gt;2009-12-18&lt;/ReleaseDate&gt; &lt;Studio&gt;Image Entertainment&lt;/Studio&gt; &lt;Title&gt;Avatar&lt;/Title&gt; &lt;/ItemAttributes&gt; &lt;/Item&gt;</pre>	<pre>{   "Title": "Avatar",   "Year": "2009",   "Rated": "PG-13",   "Released": "18 Dec 2009",   "Runtime": "162 min",   "Genre": "Action, Adventure, Fantasy",   "Director": "James Cameron",   "Writer": "James Cameron",   "Actors": "Sam Worthington, Zoe Saldana, Sigourney Weaver, Stephen Lang",   "Plot": "When his brother is killed in a robbery, paraplegic Marine Jake Sully...",   "Language": "English, Spanish",   "Country": "USA, UK",   "Awards": "Won 3 Oscars. Another 84 wins &amp; 106 nominations.",   "Poster": "http://ia.media-imdb.com/images/M/M15BM15BanBnXkFtaw00c5MTUwMw_V1_SX300.jpg",   "Metascore": "83",   "IMDbRating": "7.9",   "IMDbVotes": "818,467",   "IMDbID": "tt0499549",   "Type": "movie",   "Response": "True" }</pre>
----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------	-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

Launch Task

Processing awards data for each movie record...

TASK 3 - Recommending Movies

## Step 3: Saving the data to MySQL database

Course: Parallel Data Processing in MapReduce
Dev
Nasir
Pavitra
Siddharthan

### Amazon Movie Review Analysis

TASK 1 - Top 5 Useful Reviewers

TASK 2 - Data Feature Enhancement

In this task, we are going to use the "product/productId" field to first identify the product details by passing the ID to the Amazon Product Advertising API. So the product details would be given in the following format. Then by passing the "Title" and "ReleaseDate" field to the OMDb API we can get the information related to the movie. The data returned is in the following format:

<pre>&lt;Item&gt; &lt;ASIN&gt;B000A2XB9U&lt;/ASIN&gt; &lt;ItemAttributes&gt; &lt;Director&gt; James Cameron &lt;/Director&gt; &lt;EAN&gt;0014381273229&lt;/EAN&gt; &lt;Format&gt;Color&lt;/Format&gt; &lt;Language&gt; &lt;Name&gt;English&lt;/Name&gt; &lt;Type&gt;Original Language&lt;/Type&gt; &lt;/Language&gt; &lt;ListPrice&gt; &lt;CurrencyCode&gt;USD&lt;/CurrencyCode&gt; &lt;FormattedPrice&gt;\$19.99&lt;/FormattedPrice&gt; &lt;/ListPrice&gt; &lt;NumberOfItems&gt;1&lt;/NumberOfItems&gt; &lt;ProductGroup&gt;DVD&lt;/ProductGroup&gt; &lt;ReleaseDate&gt;2009-12-18&lt;/ReleaseDate&gt; &lt;Studio&gt;Image Entertainment&lt;/Studio&gt; &lt;Title&gt;Avatar&lt;/Title&gt; &lt;/ItemAttributes&gt; &lt;/Item&gt;</pre>	<pre>{   "Title": "Avatar",   "Year": "2009",   "Rated": "PG-13",   "Released": "18 Dec 2009",   "Runtime": "162 min",   "Genre": "Action, Adventure, Fantasy",   "Director": "James Cameron",   "Writer": "James Cameron",   "Actors": "Sam Worthington, Zoe Saldana, Sigourney Weaver, Stephen Lang",   "Plot": "When his brother is killed in a robbery, paraplegic Marine Jake Sully...",   "Language": "English, Spanish",   "Country": "USA, UK",   "Awards": "Won 3 Oscars. Another 84 wins &amp; 106 nominations.",   "Poster": "http://ia.media-imdb.com/images/M/M15BM15BanBnXkFtCw00c5MTUwMw._V1_SX300.jpg",   "Metascore": "83",   "imdbRating": "7.9",   "imdbVotes": "818,467",   "imdbID": "tt0499549",   "Type": "movie",   "Response": "True" }</pre>
----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------	------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

Launch Task
Data Successfully Stored into database

TASK 3 - Recommending Movies

## TASK 4: Recommending Movies

This task makes use of the above mentioned web interface, through which user can select options to see top 10 movies in each Genre that:

- won Oscars and other awards.
- has high IMDB ratings.
- has high amazon rating.

The joined and filtered data using SparkSQL is bulk loaded into the MySQL database. So the data is served from MySQL database, which allows faster response for a given query. The data was precomputed and stored in the database, and we used SQL commands to retrieve the data based on the user input.

Initially we identified only these two options. But as we had calculated the Amazon Movie Review Score in the second task, we thought of providing an option to filter the records based on the Review Score.



## Amazon Movie Review Analysis

TASK 1 - Top 5 Useful Reviewers

TASK 2 - Data Feature Enhancement

TASK 3 - Recommending Movies

The preprocessed data will be available in the database. So the data is served from SparkSQL database, which allows faster response for a given query. The data would be precomputed and stored in the database, and we would use SparkSQL commands to retrieve the data based on the user input.

Select one of the options below:

☐ IMDB Rating  
☐ Amazon Review Score  
☒ Oscar / Other Awards

Select one/more genres:

Action  
Drama  
Romance  
Mystery  
Thriller  
Comedy  
Fantasy  
Crime


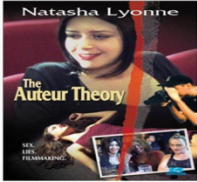

Launch Task

The output of which is the following:



## Top 10 Movies By Oscars, Wins and Nominations

For Genres : Mystery | Thriller | Comedy |

Title	Genre	Amazon Score	Awards	Poster	IMDB Rating	Language	Year
The Late Show	Comedy, Mystery, Thriller	4.0	Nominated for 1 Oscar. Another 4 wins & 6 nominations.		7.1	English	1977
The Auteur Theory	Comedy, Mystery, Thriller	5.0	1 win.		4.5	English	1999
Where's Marlowe?	Comedy, Mystery, Thriller	4.6	1 win.		6.3	English	1998

This task was executed on local machine and not on AWS.

### **Conclusion:**

We have created a running web-service that recommends you movies based on the genre you choose and based on which rating you want to sort and see the result. Also features are provided so that you can see the top k useful reviewers who were helpful in amazon and whose reviews you can trust when you are purchasing a product next time based on his review.

The main challenges for this project went into data enhancement and preprocessing so that the data will be readily available for the user who uses our web interface. The major problems faced was in the crawling part where we had to correctly adhere to the politeness policy as the calls were going in parallel. Also, the OMDb site was unstable and we had to crawl multiple times to get all the movie details. Another challenge was to use the AWS SDK-toolkit for running different applications like normal map-reduce and spark-sql. Another challenge was to setup spark-sql in AWS and pass in appropriate files into the attributes text box as the log files produced by spark-sql code were not descriptive and we had to create a SSH tunnel using dynamic port forwarding and then configure proxy settings to view the website hosted on the master node. The master node contained the descriptive logs that helped us debug various issues in running spark.

Future enhancement for this project could be to do sentiment analysis over the review text content and find out the positive and negative reviews and present that to the user so that he can learn all the pros and cons of the product before making a decision.