

# CS 6240: Parallel Data Processing

## Project Final Report

### Amazon Reviews Dataset Analysis

Shivani Gowrishankar	Sec 01
Suresh Balaji Moorthy	Sec 01
Vikash Ramesh	Sec 02

## Contents

Introduction .....	3
Data .....	4
Data Specifics .....	4
Information .....	4
Source .....	4
Sample .....	4
Stats .....	5
Data Upload .....	6
Motivation .....	6
Data Pre-Processing .....	6
Upload Process .....	6
Files .....	6
Performance .....	6
Technical Discussion .....	7
<b>Task 1</b> .....	7
Description .....	7
Approach .....	7
Files .....	8
Results .....	8
Interesting Observations and Uses of the results .....	10
<b>Task 2</b> .....	11
Description .....	11
Approach .....	11
Files .....	12
Results .....	12
Interesting Optimizations and Observations .....	14
<b>Task 3 – Top Brands</b> .....	15
Description, Approach, Files: .....	15
Results: .....	18
Interesting Optimizations and Observations .....	19
Conclusion .....	20
Results .....	20
Improvements .....	20
Summary .....	21

## Introduction

Internet networks increasingly support the ability for users to express opinions and publicly evaluate website content. These type of user evaluations can be used for a variety of purposes. We decided to use one such user evaluations dataset (Amazon reviews) to analyze and derive conclusions that could help making the product better.

On Amazon, many purchase reviews are dishonest spam entries written to skew product ratings. Identifying and encouraging useful reviewers to review more would help in improving the quality of reviews. So, we decided to identify top useful reviewers across different product categories. We identify the top useful reviewers based on the helpfulness rating of each user. Finding the helpfulness of a reviewer is an important task. We have found the helpfulness, as well as the unhelpfulness of a reviewer and then found the rating of a reviewer based on these two data points. This has helped us avoid rating spam reviewers as helpful.

Trend analysis have always been a good aid in business improvement. With the large data of reviews on each product, we decided to attempt to spot a pattern in the products' ratings. The popularity of a product over a period of time (a month) is calculated using the average number of ratings received for the product during a month.

We also analyzed the top brands in every product category. Brands are rated over a scale of 1 to 5. By identifying top rated brands in each category, we can learn which brands are leading the market in each product category.

In our project, data processing is an important task and we used a technique called Multipart. It is a method of uploading huge datasets to S3 where instead of sending one large file you send several small files to s3 which are assembled there to form the same file again. We have improved this technique by using multithreading. This enabled us to upload huge datasets very quick to S3.

Since our data consists of two datasets, we joined the data using two technologies Apache Pig and MapReduce. For joining using Pig we stored the output as flat tables and for join using MapReduce we stored the output as nested data model. We learnt about the advantages of each data model while experimenting with our tasks.

## Data

### Data Specifics

#### Information

We are working with amazon reviews dataset. We obtained the dataset by contacting Julian McAuley. The data span a period of 18 years, including 143.7 million reviews spanning May 1996 - July 2014. This dataset includes reviews (ratings, text, helpfulness votes), product metadata (descriptions, category information, price, brand, and image features), and links (also viewed/also bought graphs).

The Metadata file is about 2GB in size and includes descriptions, price, sales-rank, brand info, and co-purchasing links.

The raw review data file is approximately 20GB large. The file contains some duplicate reviews, mainly due to near-identical products whose reviews Amazon merges, e.g. VHS and DVD versions of the same movie.

#### Source

J. McAuley and J. Leskovec. Hidden factors and hidden topics: understanding rating dimensions with review text. RecSys, 2013.

#### Sample

##### *Review data*

```
{
  "reviewerID": "A2SUAM1J3GNN3B",
  "asin": "0000013714",
  "reviewerName": "J. McDonald",
  "helpful": [2, 3],
  "reviewText": "I bought this for my husband who plays the piano. He is having a wonderful time playing these old hymns. The music is at times hard to read because we think the book was published for singing from more than playing from. Great purchase though!",
  "overall": 5.0,
  "summary": "Heavenly Highway Hymns",
  "unixReviewTime": 1252800000,
  "reviewTime": "09 13, 2009"
}
```

where

- reviewerID - ID of the reviewer, e.g. [A2SUAM1J3GNN3B](#)
- asin - ID of the product, e.g. [0000013714](#)
- reviewerName - name of the reviewer
- helpful - helpfulness rating of the review, e.g. 2/3
- reviewText - text of the review
- overall - rating of the product
- summary - summary of the review
- unixReviewTime - time of the review (unix time)
- reviewTime - time of the review (raw)

### Meta data

The Metadata file is about 19GB in size and includes descriptions, price, sales-rank, brand info, and co-purchasing links.

```
{
  "asin": "0000031852",
  "title": "Girls Ballet Tutu Zebra Hot Pink",
  "price": 3.17,
  "imUrl": "http://ecx.images-amazon.com/images/I/51fAmVkTbyL._SY300_.jpg",
  "related":
  {
    "also_bought": ["B00JHONN1S", "B002BZX8Z6", "B00D2K1M3O", "B008UBQZKU", "B00D103F8U",
    "B007R2RM8W"],
    "also_viewed": ["B002BZX8Z6", "B00JHONN1S", "B008F0SU0Y", "B00BFXLZ8M"],
    "bought_together": ["B002BZX8Z6"]
  },
  "salesRank": {"Toys & Games": 211836},
  "brand": "Coxlures",
  "categories": [["Sports & Outdoors", "Other Sports", "Dance"]]
}
```

where

- asin - ID of the product, e.g. [0000031852](#)
- title - name of the product
- price - price in US dollars (at time of crawl)
- imUrl - url of the product image
- related - related products (also bought, also viewed, bought together, buy after viewing)
- salesRank - sales rank information
- brand - brand name
- categories - list of categories the product belongs to

### Stats

Dataset Statistics	
Number of reviews	78,714,813
Number of users	6,643,669
Number of products	2,441,053
Users with > 50 reviews	56,772
Median no. of words per review	82
Timespan	Jun 1995 - Mar 2013

## Data Upload

We devised a faster way to upload data to S3:

### Motivation

- Uploading to S3 through the S3 management console took a lot of time
- The upload process wasn't completely utilizing our upload bandwidth
- For large files we had to ensure the connection didn't break as we would need to start over again

### Data Pre-Processing

We preprocessed the data to eliminate the fields that were not required for a tasks performed. For eg. We did not use fields like 'related' in the metadata.

### Upload Process

We used the Multipart upload provided by AWS SDK. The advantage of Multipart upload over traditional upload is that instead of uploading a large file as single part in multipart upload it is split into several parts based on predefined part size. This part size parameter is in the control of programmer. We further improved this process by multithreading the upload process such that each part is uploaded by a separate thread. The whole system was implemented in JAVA

### Files

**MultiPartMultiThreaded.java** – contains the implementation of the above technique

### Performance

The following table gives a comparison of speed of uploading a 100mb file to S3

Part Size	S3 Management Console	Multi Part	Multi Part Multi threaded	
			Threads = 5	Threads = 10
default	107s	-	-	-
5 mb	-	97	56	53
10 mb	-	101	65	61
15 mb	-	110	70	60

## Technical Discussion

### Task 1

#### Description

The main task was to find top 'n' reviewers consists two subtasks.

1. Join the review data and meta data using Pig.
2. Find the usefulness of a reviewer (reviewer score) for each reviewer in each category.

Once we have the usefulness score for all the reviewers in every product category, we sort the data and output the top 'n' reviewers.

#### Approach

##### JSON Parser and Pig Join:

We implemented a JSON Parser to first convert each loose json row to a record in pig. Once the Meta data and Review data are parsed to pig, we joined both dataset based on the key 'asin'.

##### Calculating helpfulness of a reviewer:

Each reviewer must have given one or more reviews in a product category. So it is necessary to calculate a reviewer's helpfulness score as a cumulative score of all the reviews he/she had given. Apart from calculating the helpfulness of a reviewer, we felt it is also important to consider the "unhelpfulness" of a reviewer. A reviewer's helpfulness score of 2/5 represents 2 helpful votes and 3 unhelpful votes. The formula that we used to calculate the overall rating of a reviewer is shown below.

$$\left(\frac{v}{v+m}\right) * R_H - \left(\frac{v}{v+m}\right) * R_{UH} + \left(\frac{m}{v+m}\right) * C$$

where,

v – number of rating for a reviewer

m – minimum reviews required

$R_H$  – average helpfulness of a reviewer

$R_{UH}$  - average unhelpfulness of a reviewer

C – average rating for a category

##### Sorting reviewers based on reviewer score

The final task is to sort the reviewers based on reviewer score for each category. This is done using secondary sort design. The composite key for sorting consists of category and helpfulness score. The reviewers are sorted in descending order based on helpfulness score and grouped to the reduce call by category.

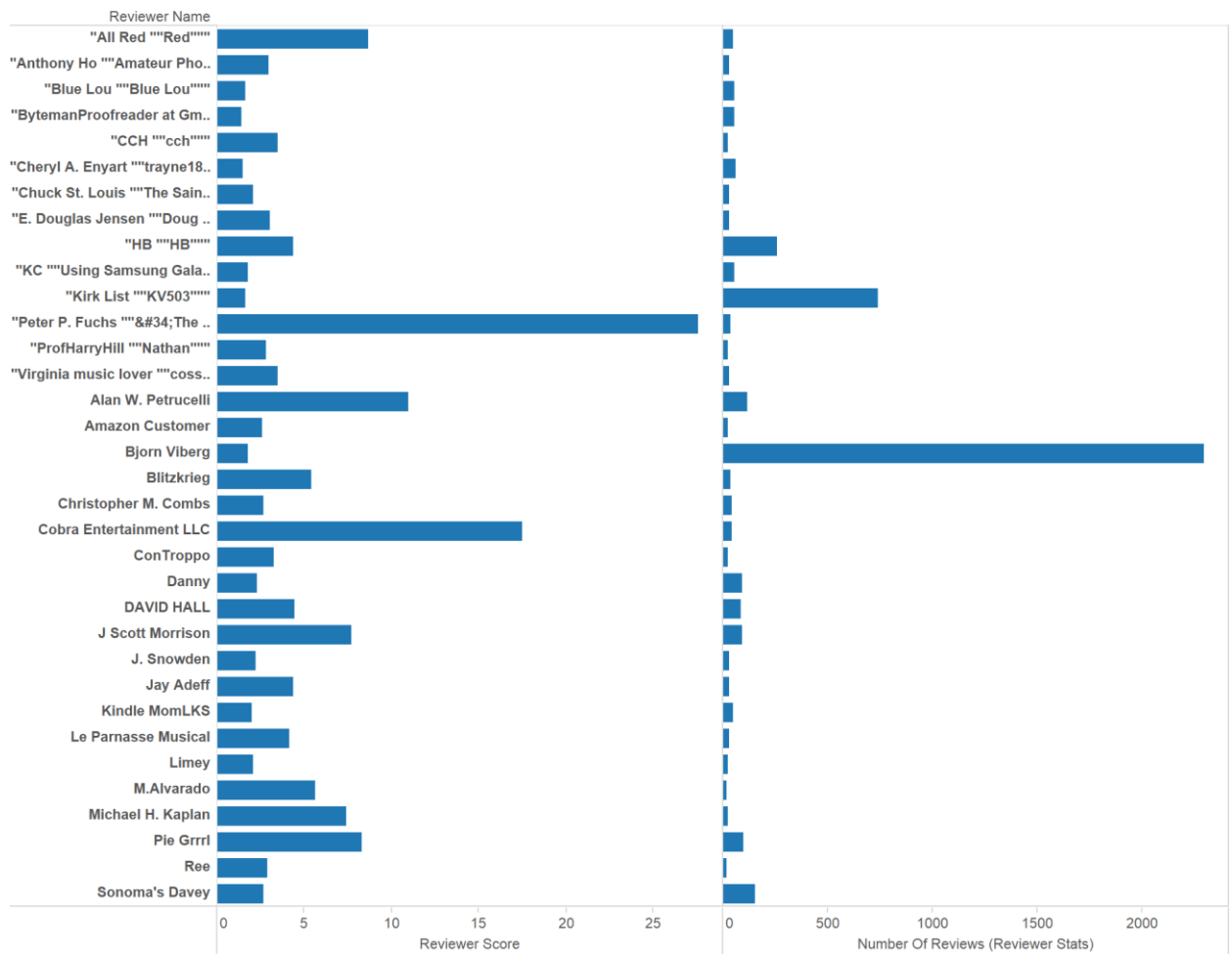
## Files

- **MetaParser.java** – Source code for meta data json parser
- **ReviewerParser.java** – Source code for reviewer data Jason parser
- **CleanData.pig** – Source code for meta data and review data join.
- **TopReviewers.java** – Source code for calculating reviewer score
- **TopFive.java** – Source code for sorting reviewers on reviewer score for each category.

## Results

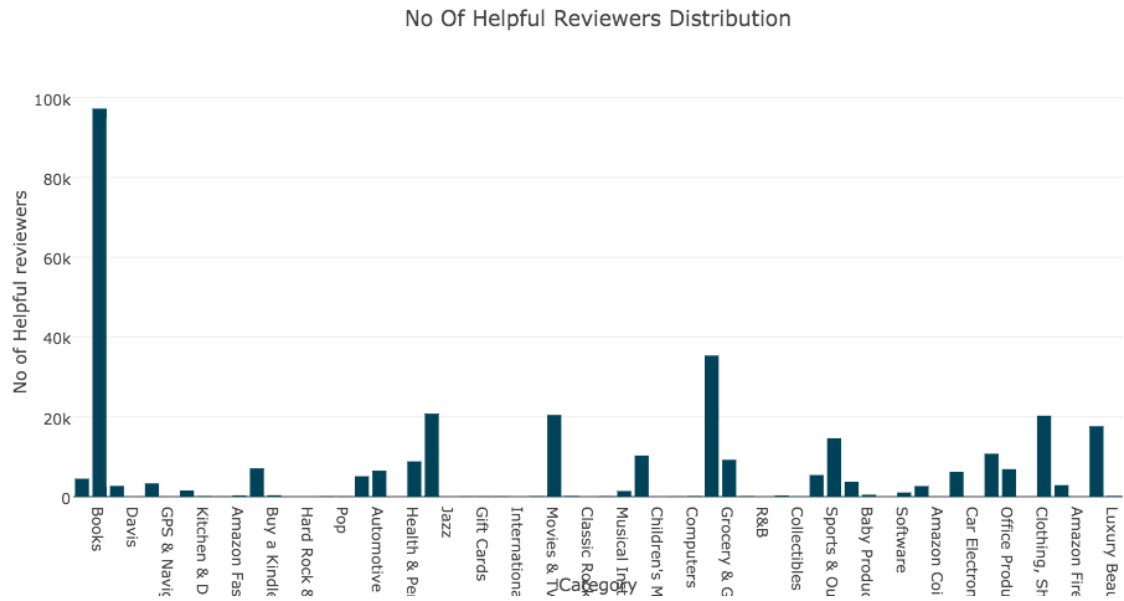
### Top reviewers in the category 'Classical'

#### Top Reviewers - Classical

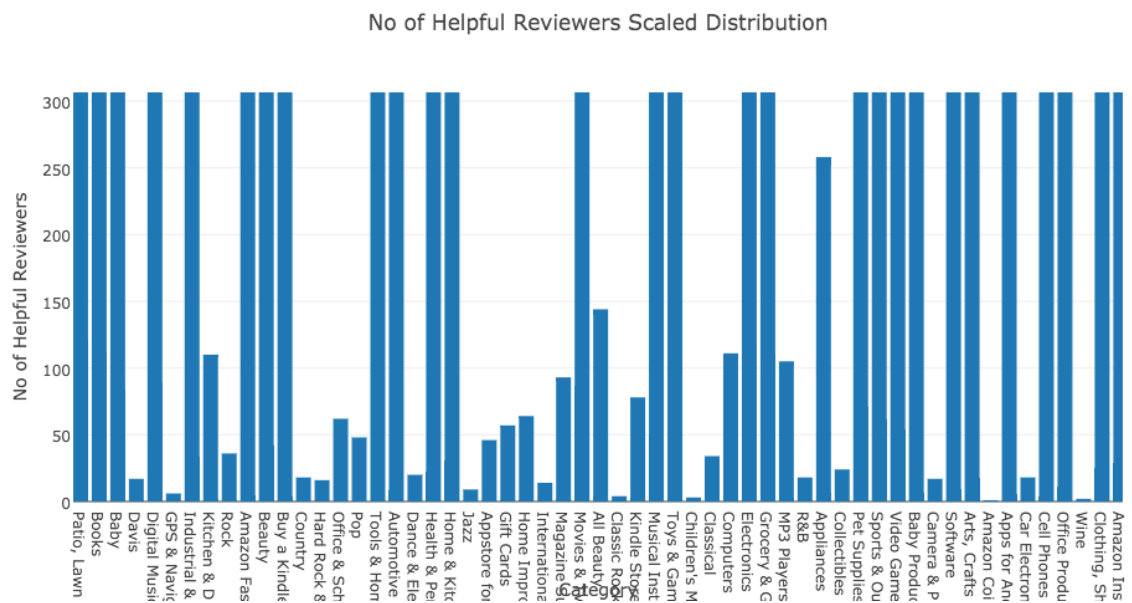




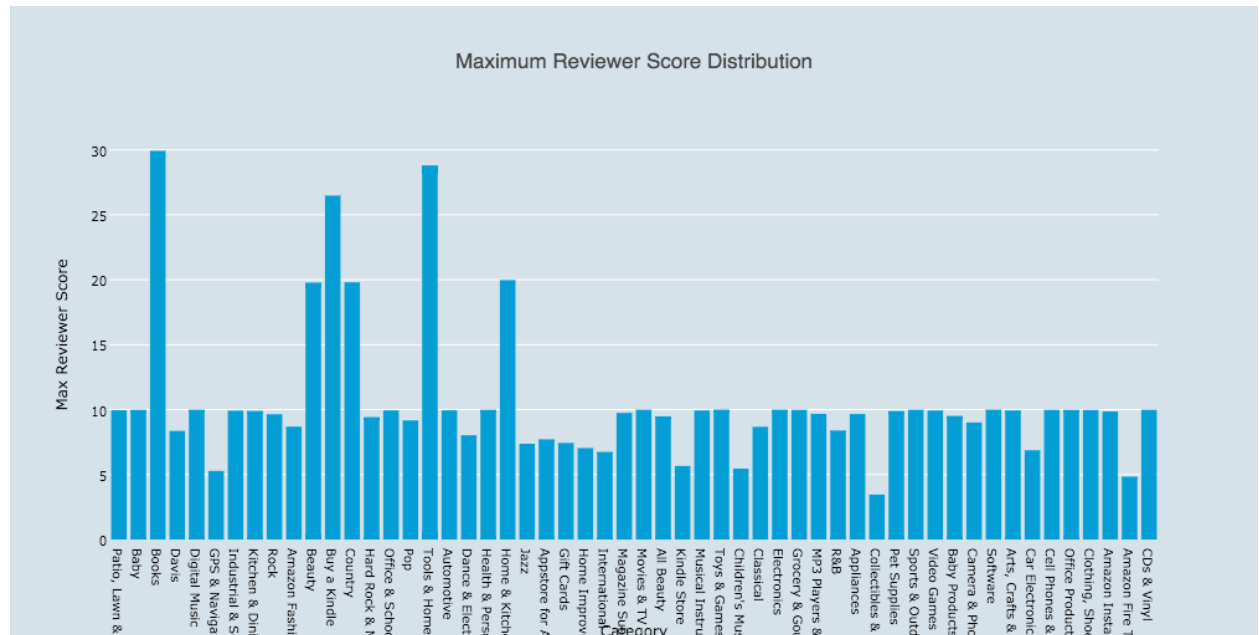
## Number of Helpful reviewers distribution



## Scaled distribution of number of helpful reviewers



### Maximum Reviewer score distribution across categories



### Interesting Observations and Uses of the results

From the results, it was interesting to observe that reviewers who had more number of review count did not have good review score. This data could be used for spam reviewer detection. We can also see that only a few categories have reviewer score in the range of 20-30. This data could be used to identify the categories that need more helpful reviewers.

## Task 2

### Description

For this task we first join the review data and metadata and use the joined output for following subtasks

1. Top 'n' products by category which have more than 'k' reviews in year range 'a-b'
2. Top 'n' products by category whose average rating decreased/increased after year 'a'
3. Year wise average rating of all products
4. Test the scalability of all these programs in different configurations

The 'n', 'k', 'a', 'b' for each program is passed as command line arguments along with the file paths to input and output directory.

### Approach

We have used nested data model for this task

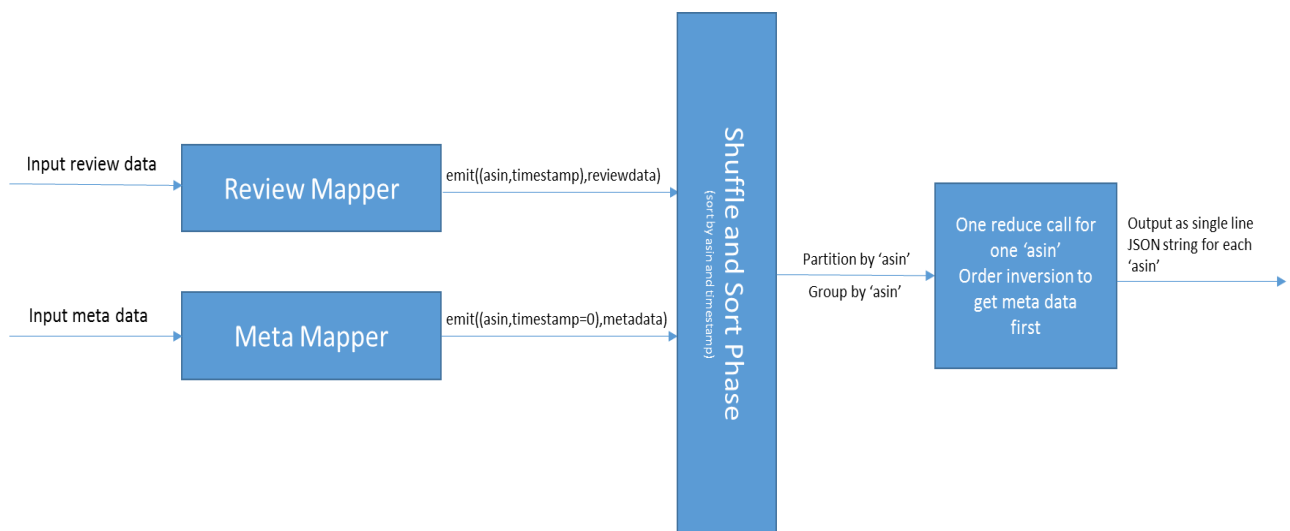
#### Nested Data Model

- Implemented using JSON
- Product id forms the parent entity which consumes all other attributes
- Advantage of nested data model is that all the subtasks are related to products ,so to calculate the statistics of a product for e.g. average rating can be calculated using a single JSON entity of that product

#### Overview of program to join the review data and metadata

- Usage of Multiple mappers to read the review data and meta data
- Usage of product id (asin) as the key and review with unix-time stamp as value
- **Order inversion** to get the product category and title
- **Partition** by 'asin' and **Group** by 'asin'
- At the end of every reduce call a product with all its reviews is written in JSON format

#### Flow diagram describing Multiple Mapper execution flow



For the subtasks '1' and subtask '2' the design pattern is the same only difference is the calculation of average rating in the Mapper.

#### Overview of program for subtask '1' and subtask '2'

- Calculate the average rating based on the given year range
- Emit this calculated parameter along with the product details as a Composite Key and the reviews within the specific range as value in JSON format
- Usage of **secondary sort design** pattern where the keys are sorted on this calculated parameter, partitioned by 'category' and finally grouped to the same reduce call based on 'category'

#### Overview of program for subtask '3'

- This program leverages on the usage of nested data model and suffices with a map only task
- One input line to the Map call contains all the reviews for a product
- Average rating for each year is calculated using memory
- Output the average rating/year for the product in the same map call

#### Files

- **Join.java** – Source code for joining review data and meta data and getting the output as nested data model
- **YearWise.java** – Source code for 3
- **ChangeTop.java** - Source code for 2
- **RangeTop.java** – Source code for 1

#### Results

##### Screenshot for successful run in AWS

Cluster: finalproj Terminated Terminated by user request

Connections: --  
Master public DNS: ec2-52-90-171-33.compute-1.amazonaws.com SSH  
Tags: --

Summary	Configuration Details	Network and Hardware	Security and Access
ID: j-26KMS6W6XTKIC Creation date: 2015-12-13 02:29 (UTC-5) End date: 2015-12-13 03:14 (UTC-5) Elapsed time: 45 minutes Auto-terminate: No Termination protection: Off	Release label: emr-4.2.0 Hadoop distribution: Amazon 2.6.0 Applications: -- Log URI: s3://aws-logs-020395049440-us-east-1/elasticmapreduce/ EMRFS consistent view: Disabled	Availability zone: us-east-1c Subnet ID: subnet-d80415a1 Master: Terminated 1 m3.xlarge Core: Terminated 15 m3.xlarge Task: --	Key name: testkey EC2 instance profile: EMR_EC2_DefaultRole EMR role: EMR_DefaultRole Visible to all users: All Change Security groups for sg-30d5c857 (ElasticMapReduce-master) Master: Security groups for Core sg-30d5c858 (ElasticMapReduce-slave) & Task:

Monitoring

Hardware

Steps

Add step Clone step

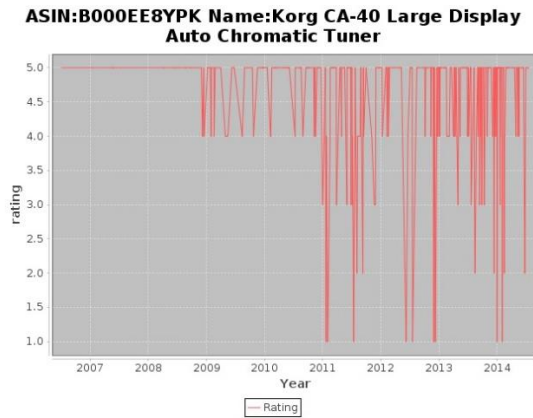
Filter: All steps Filter steps 13 steps (all loaded) View all interactive jobs View all jobs

ID	Name	Status	Start time (UTC-5)	Elapsed time	Log files	Actions
s-1DKEODRAC37B	YearWise	Completed	2015-12-13 03:11 (UTC-5)	42 seconds	controller   syslog   stderr   stdout	View jobs
s-GJ7KJZ73HND0	ChangeTop	Completed	2015-12-13 03:10 (UTC-5)	38 seconds	controller   syslog   stderr   stdout	View jobs
s-OOMJOCUEHLSK	RangeTop	Completed	2015-12-13 03:09 (UTC-5)	44 seconds	controller   syslog   stderr   stdout	View jobs
s-1XZTZNLG6R8BA	Join	Completed	2015-12-13 03:07 (UTC-5)	2 minutes	controller   syslog   stderr   stdout	View jobs
s-129KFF9SIZ540	YearWise	Completed	2015-12-13 02:57 (UTC-5)	46 seconds	View logs	View jobs
s-Z28T8U6U495WM	ChangeTop	Completed	2015-12-13 02:57 (UTC-5)	40 seconds	View logs	View jobs
s-ZUSTIH8ITSRND	RangeTop	Completed	2015-12-13 02:56 (UTC-5)	42 seconds	View logs	View jobs
s-3INVISX8LYSI	Join	Completed	2015-12-13 02:53 (UTC-5)	2 minutes	View logs	View jobs
s-6TGG6DMX3P1	YearWise	Completed	2015-12-13 02:42 (UTC-5)	54 seconds	View logs	View jobs
s-10CDCE9MCGRM	ChangeTop	Completed	2015-12-13 02:41 (UTC-5)	54 seconds	View logs	View jobs
s-39WCOVFR8BLS	RangeTop	Completed	2015-12-13 02:41 (UTC-5)	54 seconds	View logs	View jobs
s-0BF5QL74DC07	Join	Completed	2015-12-13 02:36 (UTC-5)	4 minutes	View logs	View jobs
s-2ISJUE1265FS	Setup hadoop debugging	Completed	2015-12-13 02:36 (UTC-5)	3 seconds	View logs	View jobs

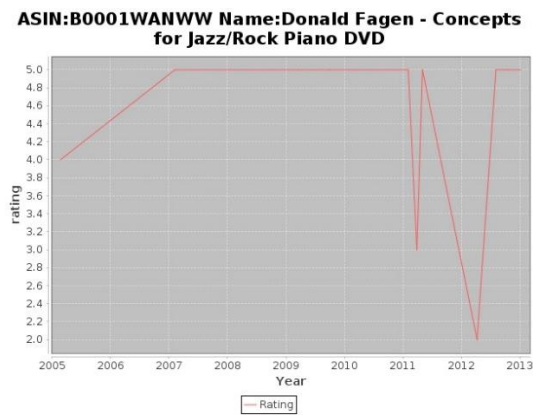
## Graphs

The output of each subtask '1', '2' and '3' is graphed.

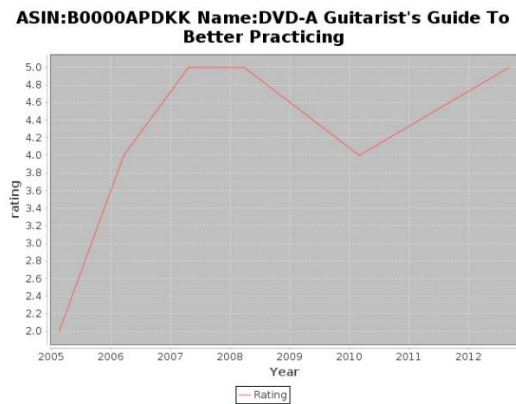
### Graph output of subtask '1' for one product



### Graph output of subtask '2' for one product



### Graph output of subtask '3' for one product



## Scalability

Machines Used: **m3.xlarge**

Cluster 1 master	JOIN	YearWise	RangeTOP	ChangeTop
5 workers	190	32	40	40
10 workers	135	30	38	36
15 workers	110	28	36	32

**Values are time taken for execution in secs on cleaned data of size – 11.06GB**

This can be verified from the line **S3: Number of bytes read=11876335529** in the corresponding syslog file of JOIN program

The execution time is taken from the last line of controller log files for corresponding tasks.

One can see from syslog files that the number of Map tasks is the same for all configurations from the syslog files.

## Interesting Optimizations and Observations

- Usage of nested data model has shown to reduce the execution time for program and in some subtasks has eliminated need for reduce phase
- Since majority of work is done map phase and shuffle and sort phase the scalability of all the programs depend on the number of map tasks which is controlled by the input size. This can be verified from the “Number of Map tasks” field in syslog files for each successful run of the program.
- Usage of multiple Mappers in the Join program enables the handling of two different datatypes by the same Map Reduce program

### Task 3 – Top Brands

Description, Approach, Files:

We wanted to find out top brands for each category using Amazon Reviews dataset. We came up with **Bayesian estimate** to calculate the weighted score for the brand in the category.

*Formula*

$$S_{(\text{brand,category})} = \frac{nr_{\text{brand}}}{(nr_{\text{brand}} + mr_{\text{brand}})} * ar_{(\text{brand,category})} + \frac{mr_{\text{brand}}}{(nr_{\text{brand}} + mr_{\text{brand}})} * ar_{\text{category}}$$

$S_{(\text{brand,category})}$  = Weighted Score for the brand in the category.

$nr_{\text{brand}}$  = number of reviews for the brand.

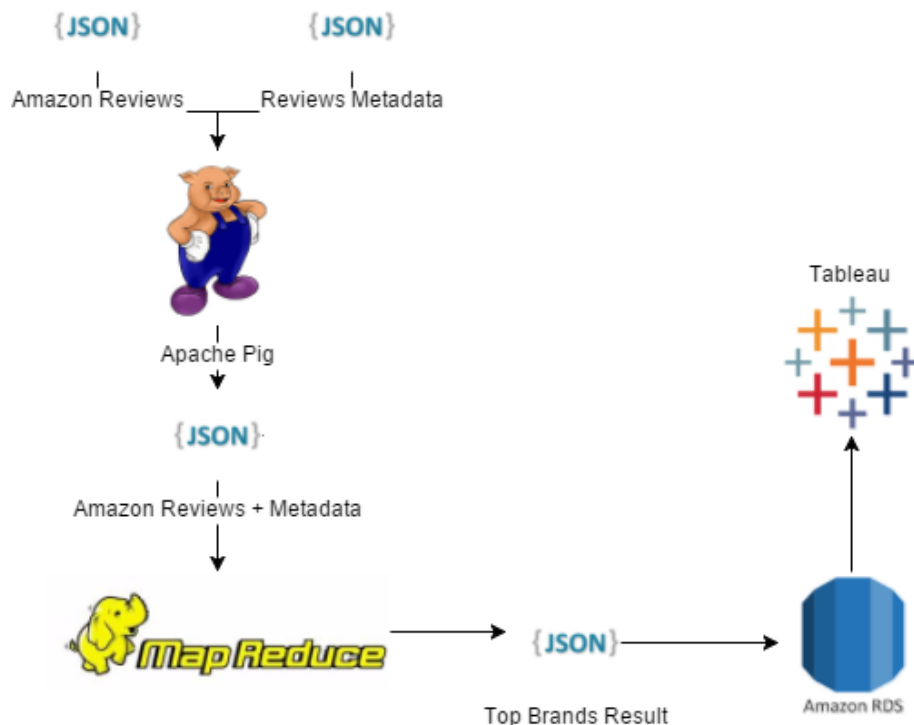
$mr_{\text{brand}}$  = minimum number of reviews for the brand.

$ar_{(\text{brand,category})}$  = average rating for the brand in the category.

$ar_{\text{category}}$  = average rating for the category.

*Architecture*

And architecture for calculating the top brands results. As an alternative architecture we also developed multi-mapper architecture replacing Apache Pig.




## Components

- **Filter Data using Apache Pig:** Used to join Amazon Reviews and Reviews Metadata using User Defined Function Parser.
  - Input Data:
    - Amazon Reviews - 18 GB or (78 Million Reviews)
      - Data Type: JSON
    - Reviews Metadata – 3 GB or (3 Million Products)
      - Data Type: JSON
  - Output Data:
    - Filtered Data – 5 GB or (78 Million Reviews with product data)
      - Data Type: CSV

**Pig File:** CleanData/CleanData.pig

**Successful Run:**

	PigCleanData	j-XJPDXGSB4L0W	Terminated All steps completed	2015-12-08 21:45 (UTC-5)	1 hour, 30 minutes	16
---	--------------	----------------	-----------------------------------	--------------------------	--------------------	----

- **User Defined Function for JSON parsing:** To process loose JSON and return as processed data to Pig.

**Project Location:**


<https://github.com/moorthys/AmazonReviews/tree/master/CleanData/CleanDataUDF>

**Project Location:** CleanData/CleanDataUDF

- **Category wise Aggregator using Pig:** Using pig we wanted to calculate average rating, helpfulness for each category.

**Pig File:** CategorywiseAggregation/CategorywiseAggregation.pig

**Successful Run:**

	CategorywiseAggreClusters	j-2PZY8PYQYS9M5	Terminated All steps completed	2015-12-09 00:30 (UTC-5)	48 minutes	8
---	---------------------------	-----------------	-----------------------------------	--------------------------	------------	---

- **Apache MapReduce:** Used to calculate top brands using the filtered data.
  - Input Data:
    - Filtered Data – 5 GB or (78 Million Reviews with product data)
      - Data Type: CSV
  - Output Data:
    - Top Brands Data – 10 MB or (4021 brands across different categories with at-least 1000 ratings)
      - Data Type: CSV

**Java Project:** TopBrands/TopBrandsAnalyzeMR

**Successful Run:**



- **Amazon RDS – MySQL:** Used to perform analytics on top of the data received.
  - Input Data:
    - Top Brands Data – 10 MB or (4021 brands across different categories with at-least 1000 ratings)
      - Data Type: CSV
  - Output Data:
    - MySQL Database helpful in various analytics.
      - Data Type: MySQL Database

**Java Project:** AWSEMRResultToMySQL/

- **Tableau**
  - Input Data:
    - Amazon RDS - MySQL Database.
      - Data Type: MySQL Database
  - Output Data:
    - Graphs using various queries.

**Tableau Public URL:**

<https://public.tableau.com/profile/publish/AmazonReviewsWorkbook/TopBrands-General#!/publish-confirm>

Results:

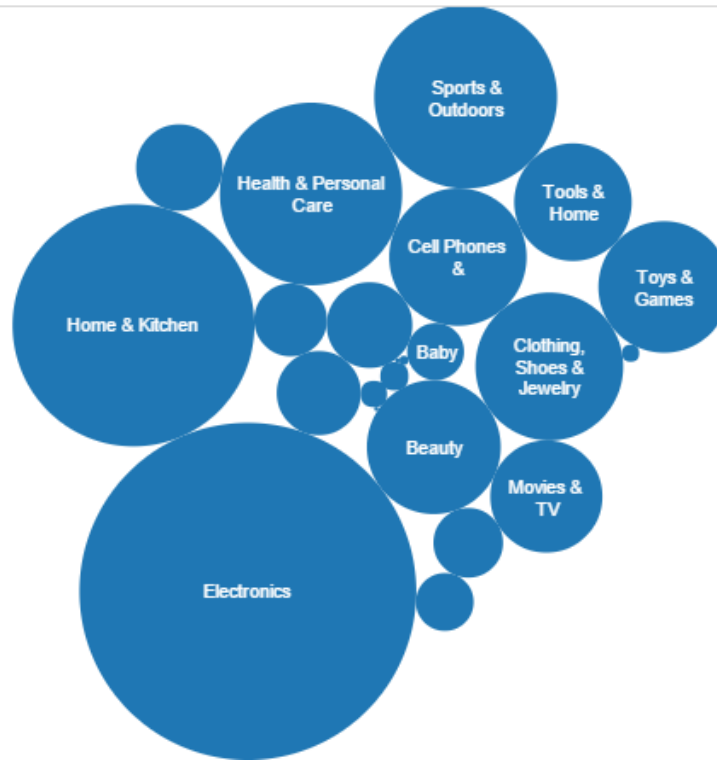


Figure 1 Category Count

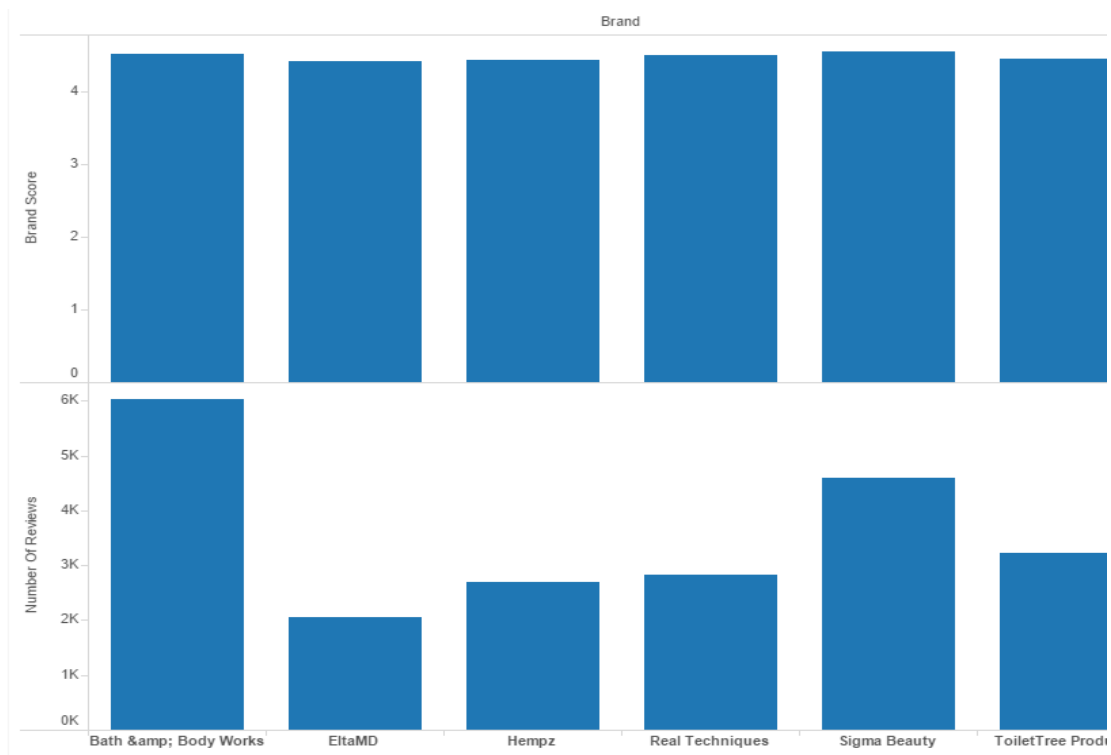


Figure 2 Top Brands - Beauty

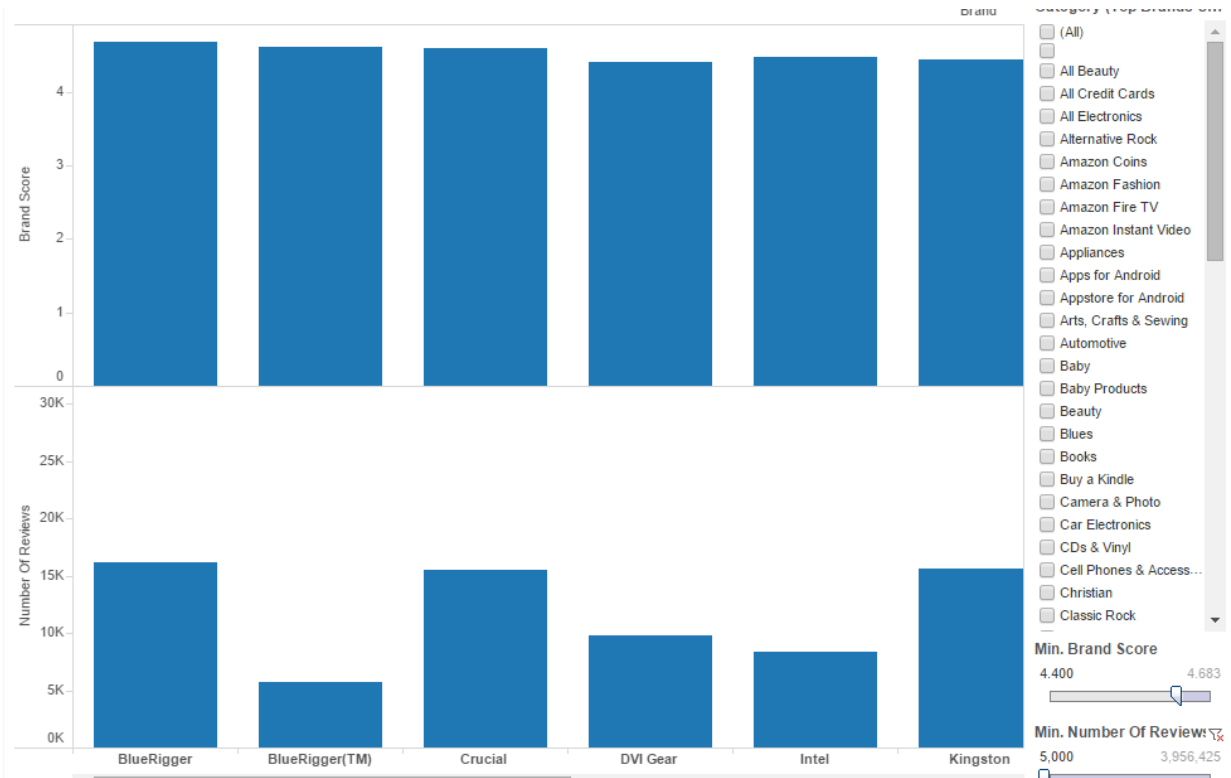


Figure 3 Top Brands – Electronics

### Interesting Optimizations and Observations

- **Observation:** Pig used one map reducer task for compressed files. Alternative is to use multi-mapper MapReduce program to read multiple data source.
- **Optimization:** Developed a control flow for execution where whenever new data is available it runs through entire architecture refreshing the data.
- **Enhancement:** Used Tableau, Amazon RDS for analytics.

## Conclusion

### Results

- Usage of Multipart Multithreaded upload to S3 improved the upload speed to S3 significantly.
- Reviewers with more reviews were not necessarily more helpful in most cases
- Nested data model performs well for product related queries.
- We cannot use the compressed file directly as input to Map Reduce program as it drastically reduces the scalability as compressed files do not have splits resulting one map task per compressed file.
- Most of the brands had same score if we consider only number of reviews so we used Bayesian estimate to include average rating of entire category which improved brand scores relatively.

### Improvements

- More faster upload to S3 possible by multithreading on Network level and also by the use of Tsunami UDP
- More complicated data model combining nested and flat structure can be used such that the model performs well for queries related to reviewers as well
- Usage of more stricter unique identifier rather than unix timestamp for each reviews.
- Task 1 could be improved by implementing a better algorithm which considers trustworthiness of a reviewer. We can also employ text based analysis to calculate the helpfulness of a reviewer.
- Top Brands can be further enhanced by adding Sentiment Analysis on review content. This factor will help brands understand how their reviews are? Positive, negative or neutral.
- We can remove duplicate reviews by checking for the review contents and performing contextual analysis
- A web UI can be built on the Amazon stack to completely automate every task as per user requirements

## Summary

- Though the input data followed JSON style format it wasn't strictly JSON as it had used single quotes instead of double quotes. So this meant built-in parsers wouldn't work. We handled this by writing our own JSON parser and adding it as user defined functions (UDF) to parse it using Pig.
- Working with data of huge size made it difficult for us to upload data to S3 using ordinary setup so we improved the speed of upload process by implementing Multithreaded Multipart Upload, which split large files into many parts and uploaded them concurrently
- Our initial algorithm calculated the cumulative helpfulness and unhelpfulness without considering the number of reviews which resulted in a poor distribution as most scores were of 0.5. We overcame this by additionally considering the average unhelpfulness and average category rating.
- Review data and Meta data each had different format so joining them in same MapReduce job was achieved using the implementation of Multiple Mappers where each Mapper read file from a separate input path. This allowed each Mapper work independently on each type of data and combine them all in Reduce Phase
- Usage of Tableau, Amazon RDS for data analytics. We used Tableau and Jfree graph creator to make the output of our tasks more readable and easily understandable for data analytics by presenting them as graphs and charts