MapReduce performance evaluation on EC2 (HDFS HA) vs EMR Application - Online spam reviewers detection

CSC 591(002) Data Intensive Computing Fall '17

Team 3

Divya Guttikonda (dguttik), Nithya Kumar (nkumar8), Sahithi Guddeti (sguddet)

Abstract

Online product reviews are becoming prevalent due to the veracity of reviews provided by the users. This particularly helps a lot of users in their decision-making process during product purchase, for example Amazon product reviews. One major problem that exists is 'opinion spamming' where fraudulent reviewers write manipulative spam reviews to promote or demote a product. Since this is a large scale distributed problem, analyzing and handling huge volumes of data is limited by storage and cost constraints. In this project, we have dealt with the above-mentioned problem by proposing a MapReduce application framework to detect online spam reviewers and by building & comparing two infrastructure models viz., Hadoop HA on EC2 (with EBS) and EMR (with S3) to handle huge volumes of input data. Our project successfully detects spammers on different product reviews big datasets [1] and also evaluates the performance of MapReduce on the built EC2 and EMR clusters. The performance evaluation is done considering the aspects of application execution time, instance costs, storage costs, number of input splits, number of map and reduce tasks, and the number of output file partitions. Our conclusions are focused on the optimality and cost-effectiveness of our MapReduce application in detecting spammers and on the infrastructure decision to be taken under different data-intensive scenarios.

I Introduction

Traditional commerce has been replaced by e-commerce by a large extent recently. One of the main reasons e-commerce is thriving is because of the ability to share feedback on the products in the form of reviews. Since there are no constraints on posting reviews, a lot of spam reviews can be found. Automation of spam review detection has been explored well in the past through natural language processing and machine learning techniques. This project aims at improving the detection process by detecting the spam reviewers rather than spam reviews. Detecting spam reviewers helps to eliminate a large number of spam reviews at once across the entire e-commerce site. This application uses the helpful attribute of the reviewers in a MapReduce job to detect spam reviewers.

Apart from the application itself, the choice of infrastructure is important to handle huge volumes of data. The MapReduce framework can be executed on different infrastructures. Each environment is optimized for specific features of application. This project focusses on investigating two such infrastructures. One is custom Hadoop cluster over EC2 instances, the other environment is AWS Elastic MapReduce with Simple Storage Service(S3). The juxtaposition of these methods brings out how they work with varying configurations and with increasing volumes.

This paper describes the details on the aforementioned ideas. Section II recounts the developments in the field of spam detection. It contains the literature review of the papers solving the mentioned problem published over the past years. Section III contains detailed explanation of the Spam detection application workflow and the architectures related to both the environments. Section IV list the results and derivations obtained. This concentrates on the evaluation of the infrastructures used. Section V describes the performance evaluation of the proposed infrastructures. Section VI determines which infrastructure is best suited for the proposed system and for different user requirements.

II Related Work

Many studies have been done in the area of spam detection. Spam detection generally uses different natural language processing and machine learning techniques. A recent paper by Chavan *et al.* (2017) [2] concentrated on spam detection using Hadoop. This paper used Parts of Speech tagging and sentiment analysis on a simple Hadoop MapReduce framework to detect spam reviews. They used three phases of map and reduce for different operations. The first one for cleaning the data, second one for finding sentiment scores for the reviews and the final one for detecting spam reviews.

Earlier papers that worked on spam detection had different ideas. They did not consider Hadoop for the computation. A famous paper by Jindal and Liu (2007) [3] solved the problem of spam detection by duplicate deletion and a classification system using probability likelihood assigned to each review. A couple of other surveys on the spam review detection systems by Crawford *et al.* (2015) [4] and Heydari *et al.* (2015) [5] discuss the various ML techniques explained in high level. But all these works do not detect spam reviewers, which if detected can be used to automatically detect spam reviews without complex analysis. This is one major motivation of our project.

Then comes the details of work performed using Hadoop and HDFS [6]. In common Hadoop architecture, NameNode was a single point of failure (SPOF) in an HDFS cluster. Each cluster had a single NameNode, and if that machine or process became unavailable, the cluster as a whole would be unavailable until the NameNode was either restarted or brought up on a separate machine. This impacted the total availability of the HDFS cluster in two major ways:

- In the case of an unplanned event such as a machine crash, the cluster would be unavailable until an operator restarted the NameNode
- Planned maintenance events such as software or hardware upgrades on the NameNode machine would result in windows of cluster downtime

In our project, the HDFS High Availability feature addresses the above problems by providing the option of running two redundant NameNodes in the same cluster in an Active/Passive configuration with a hot standby. This allows a fast failover to a new NameNode in the case that a machine crashes, or a graceful administrator-initiated failover for planned maintenance.

III Design & Implementation

This section describes the design and implementation of both the MapReduce application logic and the infrastructure of Hadoop HA cluster on Amazon EC2 & Amazon EMR.

a) Application Workflow

Spam reviewers detection application will take amazon product reviews dataset[7] stored in HDFS or S3 and runs mapper, reducer jobs in respective hadoop clusters. Initially, PoCs for detecting spam reviews were done using OpenNLP and sentiment analysis. Later on through discussion, it was concluded that it will be useful to detect spammers rather than their reviews. For this, we considered the helpful attribute as the base element which accurately identifies the number of users who found a particular review helpful. This attribute helps in differentiating spammers from genuine users due to the fact that peer users' opinions are more reliable than OpenNLP and ML techniques. On an average scenario, if many people do not a reviewer's review helpful, the reviewer has not given a genuine review. The high-level workflow of the application is given in Fig. 1.

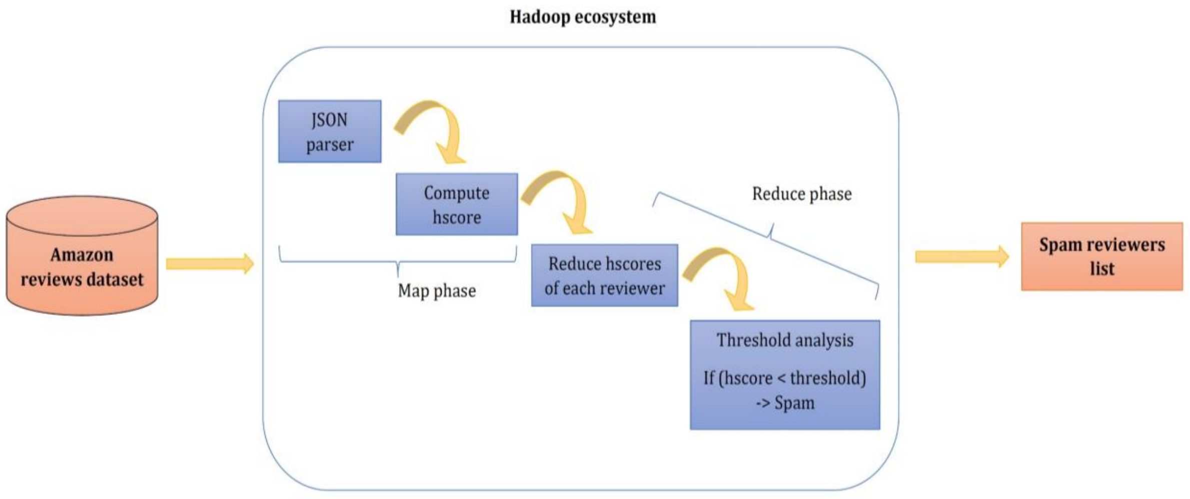


Fig. 1 High-level workflow of MapReduce application

The MapReduce phases of the application are detailed as follows. Fig. 2 shows the implemented MapReduce framework.

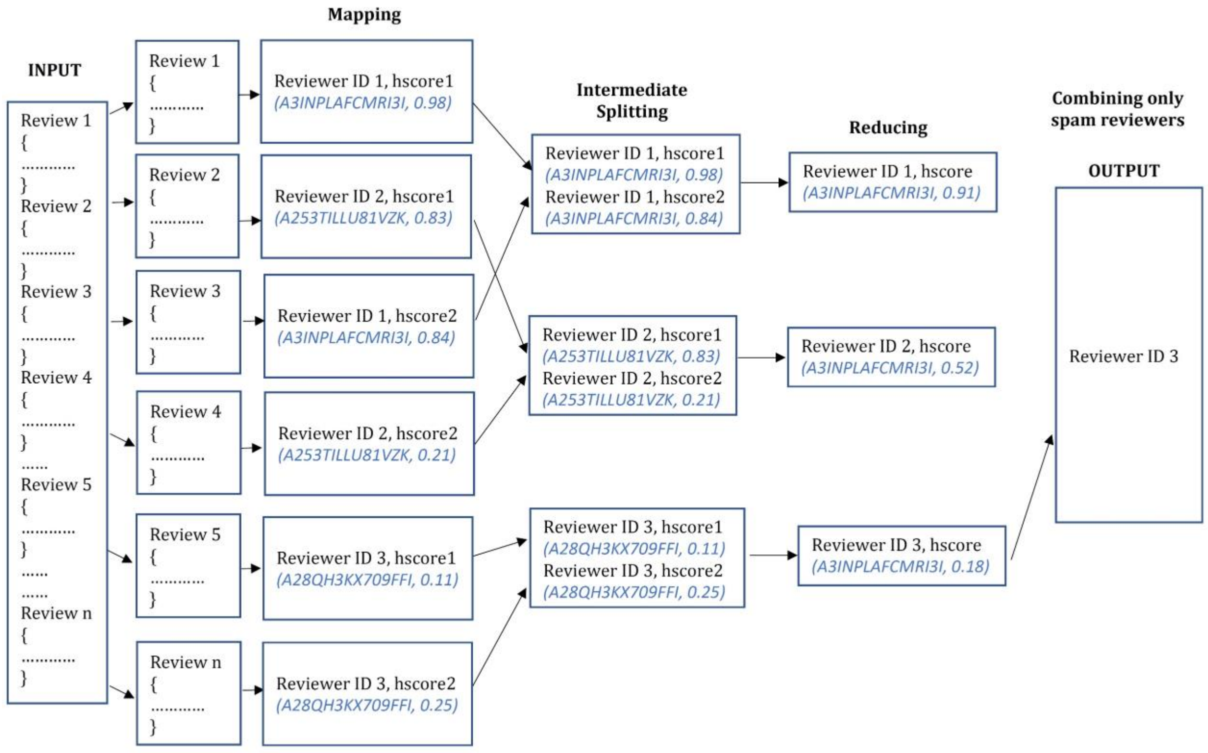


Fig. 2 MapReduce framework

Map phase

The input data is given in JSON format to Mapper. JSON parser converts the JSON records to Text format and iterates over them and stores the “Reviewer_ID” , “helpful” attribute values (h_score) as Key, Value pair respectively in map job. These Map jobs run parallely and send the output to reduce jobs.

Reduce phase

Reducer adds up the h_scores of records grouped by reviewers and calculates an average h_score value of each reviewer. This final h_score value is then compared with predefined threshold value “limit”. If h_score of reviewer is less than threshold, then the reviewer is declared as spammer and added to reduce output file.

b) Infrastructure

In our work, the application built for detecting spam reviewers was implemented on two MR infrastructures viz., Hadoop HA cluster over AWS EC2 with EBS and AWS EMR with S3. We had built the former infrastructure from scratch on EC2 considering High Availability of the system. Both the infrastructures are discussed in the subsequent sections.

The hardware specifications for these infrastructures are,

EC2 instance: c4.4xlarge VCPU: 16, ECU: 62, Memory: 30 GB, Instance storage: EBS only

EBS Volume: 200 GB (General Purpose SSD with GP2)

Case 1: Hadoop HA cluster over AWS EC2 and EBS [8][9]

The architecture, instance specifications and storage for building Hadoop HA cluster over AWS EC2 and EBS are as follows.

Architecture

We have configured and implemented the HDFS HA using the Quorum Journal Manager (QJM) to share edit logs between the Active and Standby NameNodes. The high-level architecture of the HA infrastructure can be seen in Fig. 3.

For High Availability

In our project, we have setup the Hadoop cluster with two separate machines are configured as NameNodes. At any point in time, exactly one of the NameNodes is in an Active state, and the other is in a Standby state. The Active NameNode is responsible for all client operations in the cluster, while the Standby is simply acting as a slave, maintaining enough state to provide a fast failover if necessary.

In order for the Standby node to keep its state synchronized with the Active node, both nodes communicate with a group of separate daemons called "JournalNodes" (JNs). When any namespace modification is performed by the Active node, it durably logs a record of the modification to a majority of these JNs. The Standby node is capable of reading the edits from the JNs, and is constantly watching them for changes to the edit log. As the Standby Node sees the edits, it applies them to its own namespace. In the event of a failover, the Standby will ensure that it has read all of the edits from the Journal Nodes before promoting itself to the Active state. This ensures that the namespace state is fully synchronized before a failover occurs.

In order to provide a fast failover, it is also necessary that the Standby node have up-to-date information regarding the location of blocks in the cluster. In order to achieve this, the DataNodes are configured with the location of both NameNodes, and send block location information and heartbeats to both. In our architecture, the DataNodes have their storage distributed on Amazon EBS.

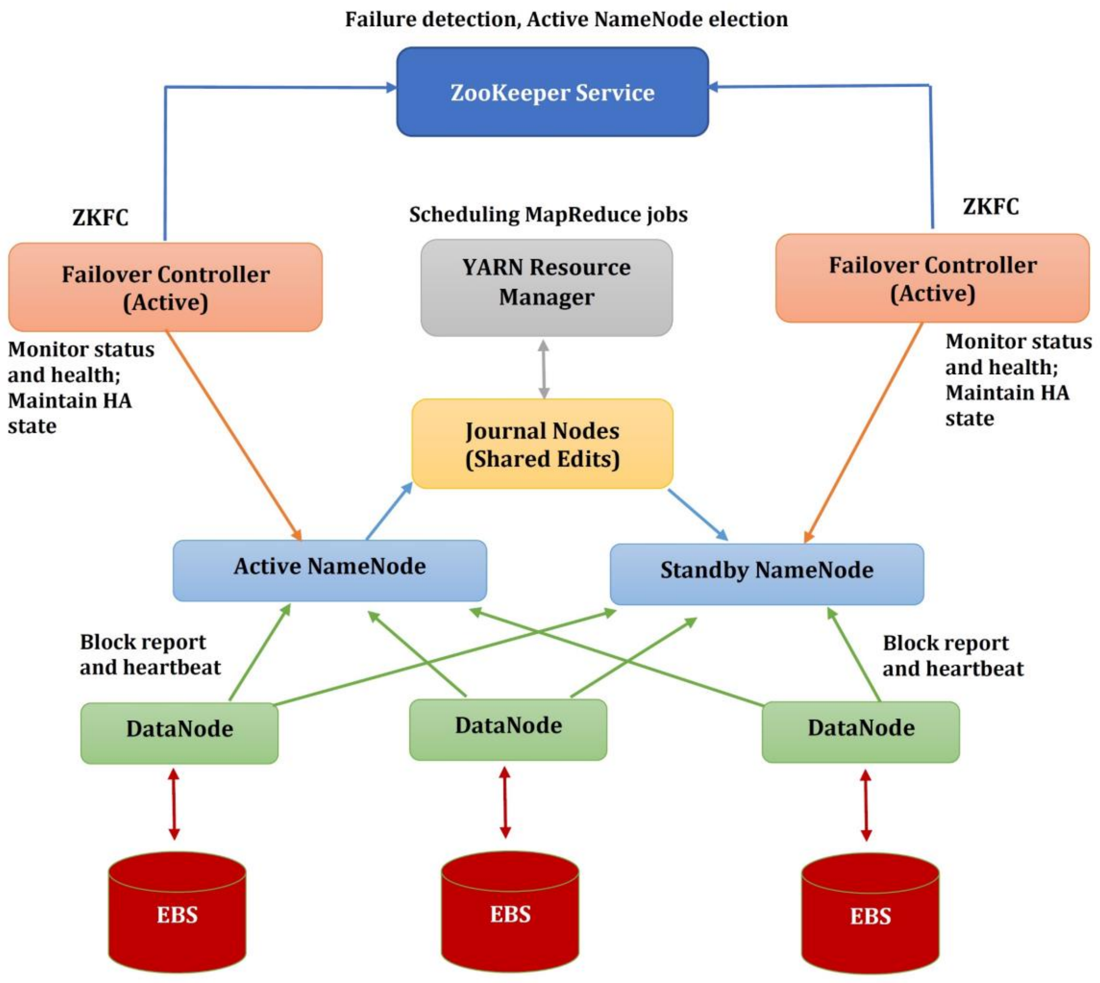

Failure detection, Active NameNode election
ZooKeeper Service
ZKFC
Scheduling MapReduce jobs
ZKFC
Failover Controller (Active)
YARN Resource Manager
Failover Controller (Active)
Monitor status and health; Maintain HA state
Monitor status and health; Maintain HA state
Journal Nodes (Shared Edits)
Active NameNode
Standby NameNode
Block report and heartbeat
Block report and heartbeat
DataNode
DataNode
DataNode
EBS
EBS
EBS

Fig. 3 Hadoop HA cluster over AWS EC2 and EBS

For Automatic Failover

To support automatic failover, we have configured two new components to the above HDFS deployment: a ZooKeeper quorum, and the ZKFailoverController process (abbreviated as ZKFC).

Apache ZooKeeper [10] is a highly available service for maintaining small amounts of coordination data, notifying clients of changes in that data, and monitoring clients for failures. The implementation of automatic HDFS failover relies on ZooKeeper for the following things:

1. **Failure detection** - each of the NameNode machines in the cluster maintains a persistent session in ZooKeeper. If the machine crashes, the ZooKeeper session will expire, notifying the other NameNode that a failover should be triggered.

2. **Active NameNode election** - ZooKeeper provides a simple mechanism to exclusively elect a node as active. If the current active NameNode crashes, another node may take a special exclusive lock in ZooKeeper indicating that it should become the next active.

The ZKFailoverController (ZKFC) [11] is a new component which is a ZooKeeper client which also monitors and manages the state of the NameNode. Each of the machines which runs a NameNode also runs a ZKFC, and that ZKFC is responsible for:

1. **Health monitoring** - the ZKFC pings its local NameNode on a periodic basis with a health-check command. So long as the NameNode responds in a timely fashion with a healthy status, the ZKFC considers the node healthy. If the node has crashed, frozen, or otherwise entered an unhealthy state, the health monitor will mark it as unhealthy.

2. **ZooKeeper session management** - when the local NameNode is healthy, the ZKFC holds a session open in ZooKeeper. If the local NameNode is active, it also holds a special "lock" znode. This lock uses ZooKeeper's support for "ephemeral" nodes; if the session expires, the lock node will be automatically deleted.

3. **ZooKeeper-based election** - if the local NameNode is healthy, and the ZKFC sees that no other node currently holds the lock znode, it will itself try to acquire the lock. If it succeeds, then it has "won the election", and is responsible for running a failover to make its local NameNode active. The failover process is similar to the manual failover described above: first, the previous active is fenced if necessary, and then the local NameNode transitions to active state.

Instance specifications

We have configured the HA cluster with the following instances each running the respective set of hadoop daemons.

1. NameNode 1 (Active NameNode) - NameNode daemon , ZKFC daemon, ZK Quorum

2. NameNode 2 (Standby NameNode) - NameNode daemon, ZKFC daemon, ZK Quorum

3. JournalNode - JournalNode daemon, Resource Manager daemon, Node Manager daemon, ZK Quorum, Map Reduce Job History Server
4. Data Node 1 - DataNode daemon, ZK Quorum
5. Data Node 2 - DataNode daemon, ZK Quorum
6. Data Node 3 - DataNode daemon, ZK Quorum

Storage

Persistence storage for data nodes is provided by attaching EBS (Elastic Block Storage) Volumes(General purpose SSDs with gp2). These volumes act as regular file system and can be attached and detached to the instance nodes without disturbing the environment. Data in EBS volumes is retained even when a connected instance node is terminated or failed.

Case 2: AWS EMR cluster using S3

The architecture, storage and workflow for building and testing the MapReduce application on AWS EMR are as follows.

Architecture

AWS EMR cluster is created with one master and 2 core EC2 instances. It is by default pre-configured to provide to reliability and replaces failed instances with other nodes, so there is no need of secondary namenode to handle fault tolerant issues. Also, EMR process huge volumes by in parts of data accessed from S3 thus not overloading the nodes. So, to avoid wasting resources, less number of instances are added to cluster. There is an option to add task nodes which does not share data but run compute to handle heavy workload. Fig. 4 shows the architecture of EMR cluster.

Storage/File System

EMR supports three kinds of file system storages [12]. HDFS storage resides data in EC2 instances and is mostly used as intermediate storage to provide caching abilities. Local file system is similar to standalone hard disk storage and stores data in EC2 instances. These two file systems boost I/O performance of EMR cluster, but they do not provide persistent storage i.e, data will be lost once the EC2 instances are terminated. So, S3 is chosen to provide persistent and scalable storage. Also, S3 supports various formats of data like json, csv, txt etc. [13].

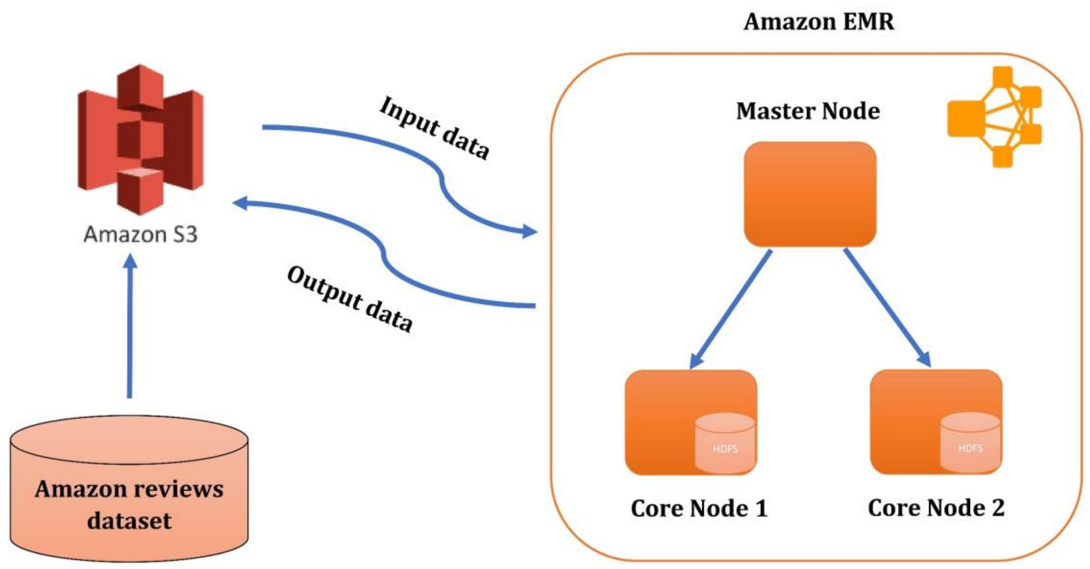


Fig. 4 AWS EMR cluster using S3

Workflow

Initially a storage bucket is created in S3 and JSON format datasets of different sizes are uploaded in it. A jar file is created with MapReduce application either externally or in master node and stored in S3. A step or job flow is created in Aws EMR UI console, which takes the application jar file and paths of input dataset and output folders stored in S3 as arguments. This job then processes the input data set and puts output files in the given folder of S3.

Fig. 5 displays this workflow.

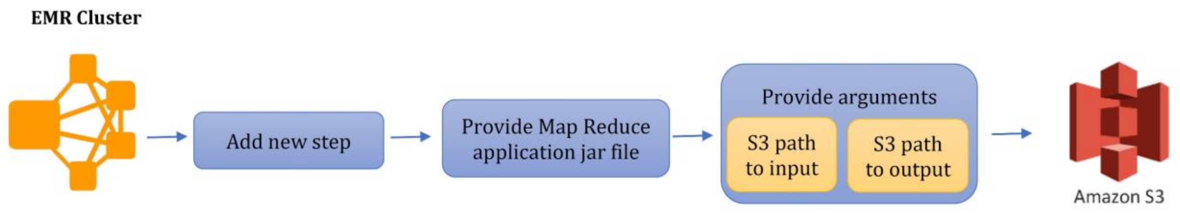


Fig. 5 AWS EMR cluster using S3

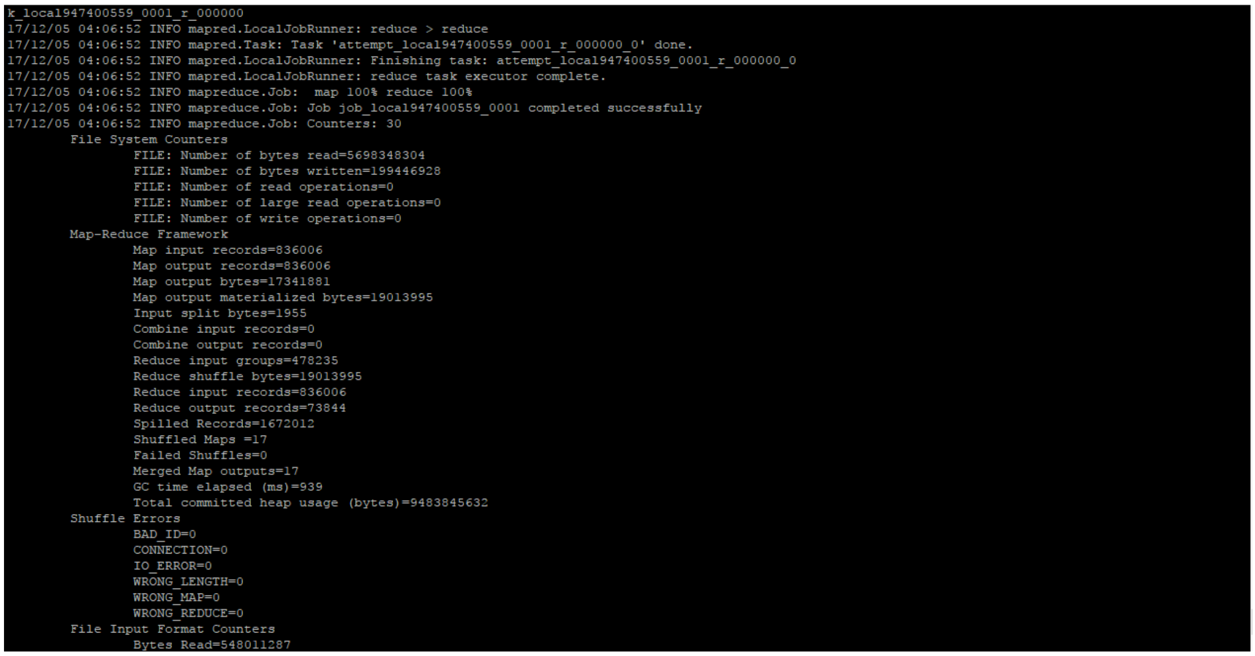
IV Results

The results of MapReduce application run on the above discussed architectures can be seen in this section.

1. Hadoop HA cluster over AWS EC2 and EBS

We configured and implemented the above-mentioned Hadoop HA cluster on EC2 instances but due to connectivity issues on AWS, we were not able to put data on Hadoop DFS. Hence, we implemented the architecture as a pseudo-distributed Hadoop cluster on a single instance. This cluster was used to test our MapReduce application as well as to evaluate performance of the infrastructure.

Fig. 6 displays the screenshot of the result of execution of the MR application on Hadoop HA.



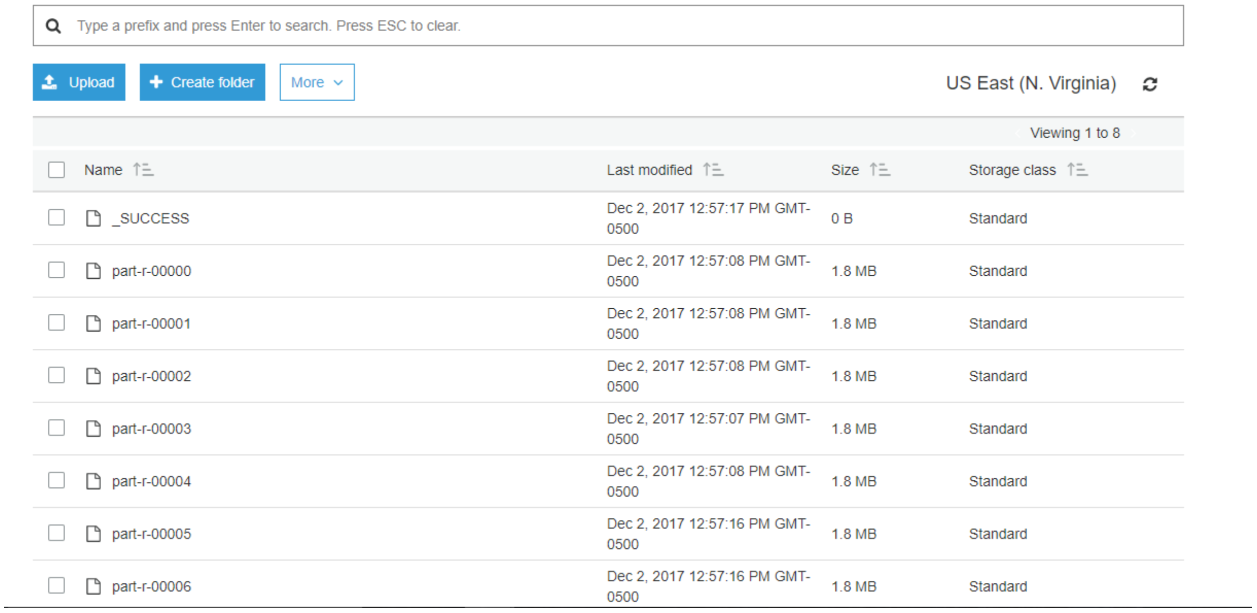
```
k_local947400559_0001_r_000000
17/12/05 04:06:52 INFO mapred.LocalJobRunner: reduce > reduce
17/12/05 04:06:52 INFO mapred.Task: Task 'attempt_local947400559_0001_r_000000_0' done.
17/12/05 04:06:52 INFO mapred.LocalJobRunner: Finishing task: attempt_local947400559_0001_r_000000_0
17/12/05 04:06:52 INFO mapred.LocalJobRunner: reduce task executor complete.
17/12/05 04:06:52 INFO mapreduce.Job:  map 100% reduce 100%
17/12/05 04:06:52 INFO mapreduce.Job: Job job_local947400559_0001 completed successfully
17/12/05 04:06:52 INFO mapreduce.Job: Counters: 30
        File System Counters
                FILE: Number of bytes read=5698348304
                FILE: Number of bytes written=199446928
                FILE: Number of read operations=0
                FILE: Number of large read operations=0
                FILE: Number of write operations=0
        Map-Reduce Framework
                Map input records=836006
                Map output records=836006
                Map output bytes=17341881
                Map output materialized bytes=19013995
                Input split bytes=1955
                Combine input records=0
                Combine output records=0
                Reduce input groups=478235
                Reduce shuffle bytes=19013995
                Reduce input records=836006
                Reduce output records=73844
                Spilled Records=1672012
                Shuffled Maps =17
                Failed Shuffles=0
                Merged Map outputs=17
                GC time elapsed (ms)=939
                Total committed heap usage (bytes)=9483845632
        Shuffle Errors
                BAD_ID=0
                CONNECTION=0
                IO_ERROR=0
                WRONG_LENGTH=0
                WRONG_MAP=0
                WRONG_REDUCE=0
        File Input Format Counters
                Bytes Read=548011287
```

Fig. 6 MR application on Hadoop HA

2. AWS EMR cluster using S3

Based on the EMR architecture and workflow mentioned above, we executed our MapReduce application on the EMR cluster.

Fig. 7 displays the screenshot of the result of execution of the MR application on EMR.


Type a prefix and press Enter to search. Press ESC to clear.
Upload
Create folder
More
US East (N. Virginia)
Viewing 1 to 8

Name	Last modified	Size	Storage class
_SUCCESS	Dec 2, 2017 12:57:17 PM GMT-0500	0 B	Standard
part-r-00000	Dec 2, 2017 12:57:08 PM GMT-0500	1.8 MB	Standard
part-r-00001	Dec 2, 2017 12:57:08 PM GMT-0500	1.8 MB	Standard
part-r-00002	Dec 2, 2017 12:57:08 PM GMT-0500	1.8 MB	Standard
part-r-00003	Dec 2, 2017 12:57:07 PM GMT-0500	1.8 MB	Standard
part-r-00004	Dec 2, 2017 12:57:08 PM GMT-0500	1.8 MB	Standard
part-r-00005	Dec 2, 2017 12:57:16 PM GMT-0500	1.8 MB	Standard
part-r-00006	Dec 2, 2017 12:57:16 PM GMT-0500	1.8 MB	Standard

Fig. 7 MR on EMR

V Performance Evaluation

The performance evaluation of the above mentioned infrastructures was done considering the aspects of application execution time, instance costs, storage costs, number of input splits, number of map and reduce tasks, and the number of output file partitions. The results and comparisons are detailed in this section.

a) Execution time

Execution time of application is less in Hadoop cluster in EC2 than EMR on S3. The reason for this is the infrastructure has directly attached compute optimised persistent EBS volumes to the data nodes. Whereas in S3, it is a distant storage server. Therefore, there is higher latency while performing IOPS operations with S3. Fig. 8 shows the execution time comparison.

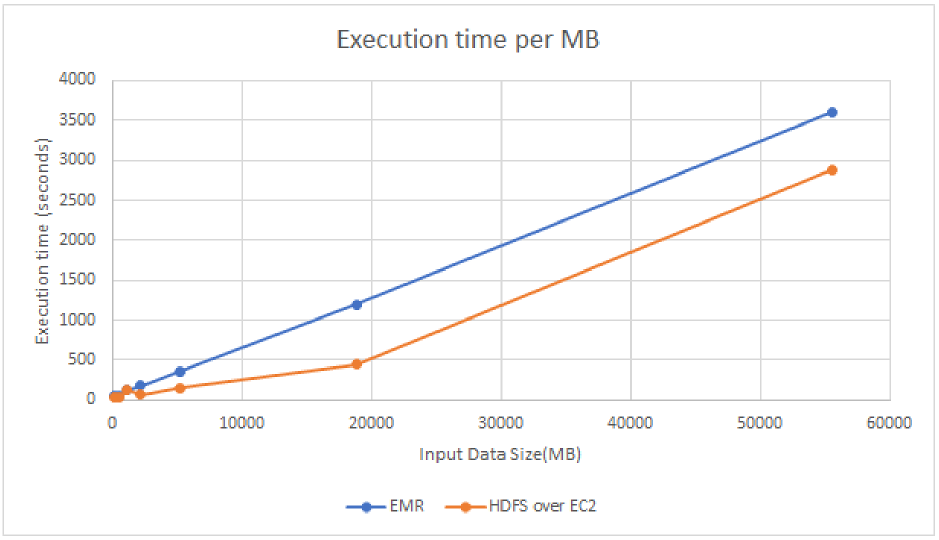


Fig. 8 Execution time comparison

b) Instance cost

As already mentioned in the above section, AWS EMR has 3 EC2 instances. But, HDFS over EC2 is implemented on single EC2 instance. The EC2 instance used in these infrastructures costs \$0.796 per hour.[14] For AWS EMR, there is an additional cost \$0.210 per hour to maintain the EMR server. So, the overall cost per hour for hadoop cluster over EC2 instances is \$0.796 per hour[15]. For AWS EMR, it is (0.796*3+0.210) = \$ 2.598 per hour. In Fig. 9, we have given the instance cost estimation for both the infrastructures. This is provided for a range that is from one hour to one month duration. Also we can notice that, EMR is more costly as it provides additional services to provide EC2 instances. There is no additional configuration from the user's side. But HDFS over EC2 needs a lot of manual configuration to set up the cluster.

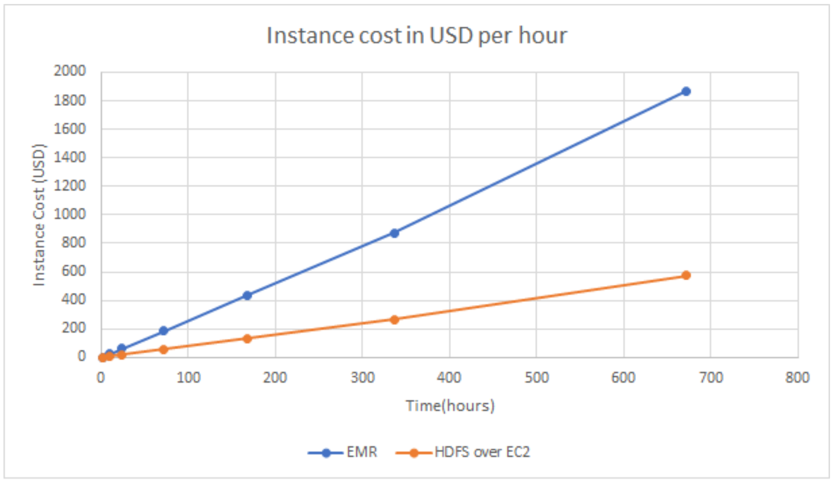


Fig. 9 Instance cost comparison

c) Storage cost

S3 standard storage costs $ 0.023 per GB. The total IOPS operations costs $0.005 per 1000 requests[16]. Around four IOPS operations are performed by 1MB storage. Therefore, the IOPS cost per GB is $0.02. The total storage cost for S3 is 0.02+0.023 = $0.043 per GB. EBS general purpose storage (GP2) costs around $0.10 per GB including IOPS provisions [17]. Fig. 10 is the cost estimation of storage ranging from 1GB to 128GB. This is the cumulative storage sum of used datasets.

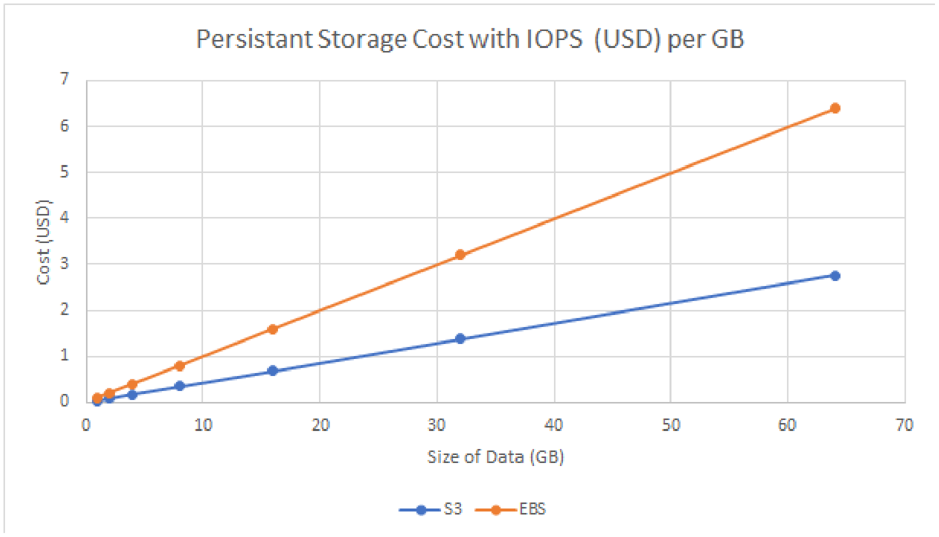


Fig. 10 Storage cost comparison

From the figure it can be inferred that S3 costs a lot cheaper than the EBS volumes. The reason being EBS volumes are IOPS optimised and directly attached to the EC2 instance. These EBS volumes are built on fast SSDs. S3 is a data server stored in a distant location from an EMR cluster. Also, S3 is made efficient to store data rather than for performing efficient IOPS operations.

d) Input splits

Whenever the data is split into small chunks. The size of these chunks is generally fixed. These chunks are then replicated in various nodes to provide fault tolerance. Referring to Fig. 11, EMR splits the data into much smaller chunks compared to Hadoop over EC2 MapReduce framework. Due to this, EMR stores the data with more reliability compared to the other infrastructure. But, Hadoop over EC2 has high consistency as data is not as replicated as it was in EMR. Additionally, these replicas are not very distant.

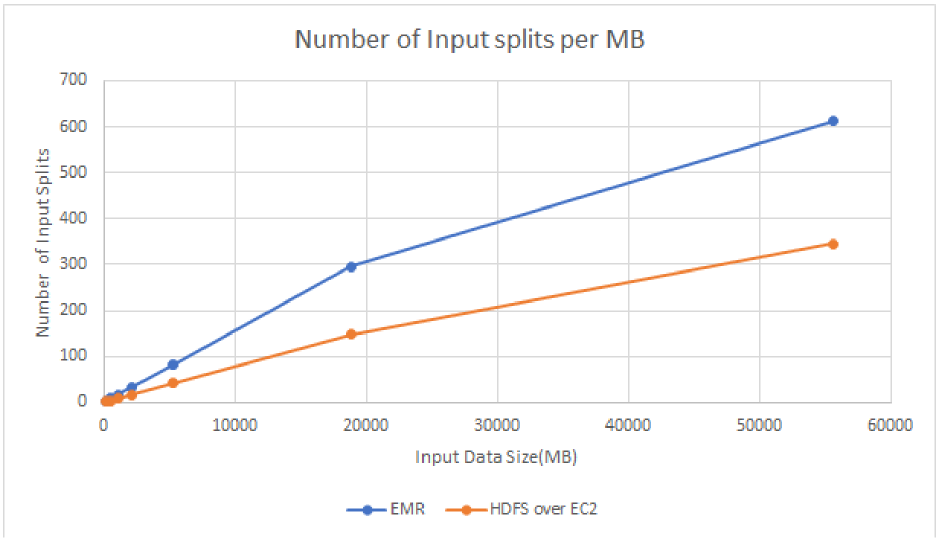


Fig. 11 Input splits comparison

e) Map and Reduce tasks

EMR has a master node and two core nodes dedicated to implement the MapReduce framework whereas Hadoop over EC2 has single node to implement the MapReduce application. So, the number of map tasks are higher in AWS EMR compared to HDFS over EC2. Fig. 12 shows reduce tasks comparison.

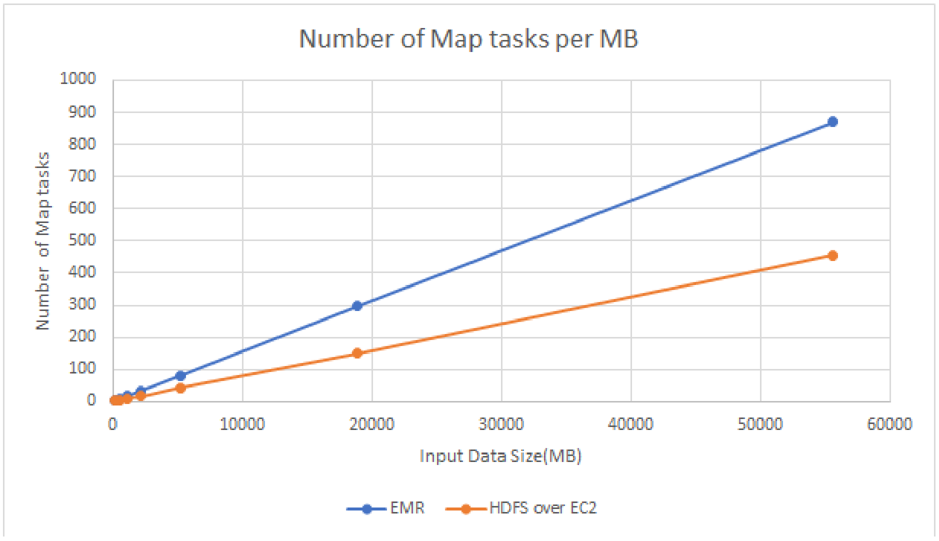


Fig. 12 Map tasks comparison

For reduce jobs, AWS EMR has more number of processor threads. Whereas, Hadoop over EC2 has processors of single EC2 instance. Because of this, higher number of reduce jobs are executed in AWS EMR compared to the other infrastructure. Fig. 13 shows reduce tasks comparison.

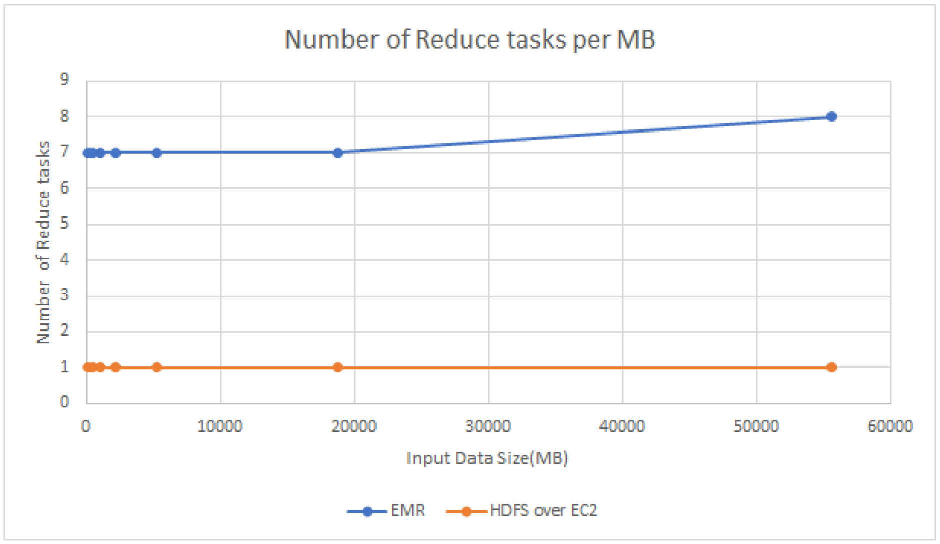


Fig. 13 Input splits comparison

f) Output parts

The number of output file parts depends on the number of reduce jobs. The results shown in Fig. 14 totally depends on the reduce tasks of respective infrastructure.

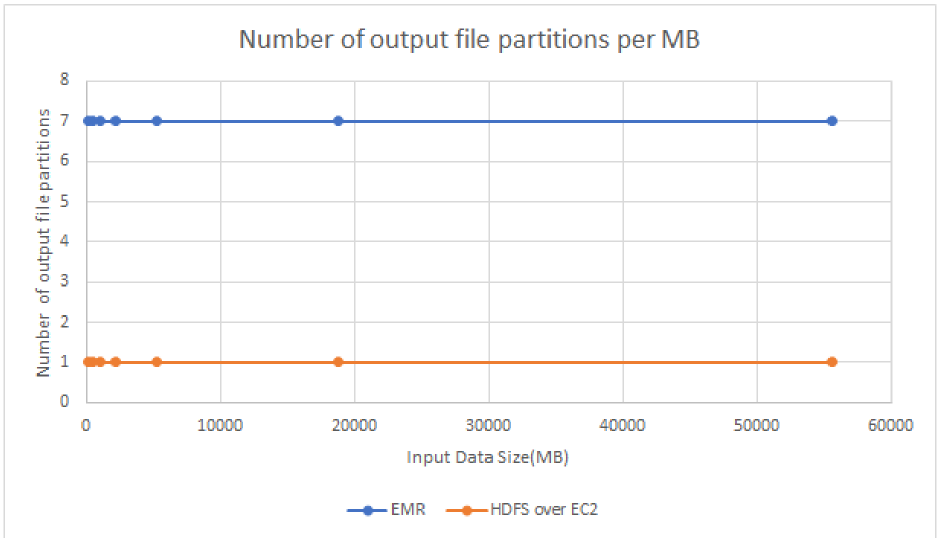


Fig. 14 Output parts comparison

VI Conclusion

The proposed application is efficient in determining the genuineness of a review by detecting spam reviewers. This approach is cost efficient compared to finding spam reviews. We assert that usage of techniques and mechanism provided by MapReduce framework significantly reduces the time complexity of application with such intensive processing. Also, the proposed infrastructures are built to handle huge amount of data approximately ranging from 100 to 200 GB.

From the performance evaluations, we conclude that HDFS cluster over EC2 is best suited for applications which are I/O intensive. It also provides high consistency but S3 is best suited if the user is not concerned with high availability of data (S3 has eventual consistency) and wants to build a scalable storage oriented application infrastructure. As our proposed system is an analytics application, it performs a lot of I/O operations. Due to this reason, we incline towards the customized Hadoop cluster over EC2 instances using EBS compared to AWS EMR using S3.

In future, there is scope for optimizing the application by using machine learning techniques. Also, the Hadoop cluster over EC2 instances can be extended to test high availability scenarios.

Acknowledgements

We would like to thank our course instructor Dr Vincent Freeh for providing us with an immense learning opportunity by virtue of this project and also for introducing us to a wide range of open-source frameworks and technologies prevalent in the data-intensive computing world. We would also like to thank Prof. Julian McAuley (Assistant Professor, Computer Science Department @ UCSD) for giving us permission to use Amazon product reviews big datasets[18] for testing our project.

References

1. http://jmcauley.ucsd.edu/data/amazon/links.html
2. http://www.ijecs.in/index.php/archive/152-volume-6-issue-2-feb-2017/2038-spam-reviews-detection-using-hadoop
3. https://dl.acm.org/citation.cfm?id=1242759
4. https://link.springer.com/article/10.1186/s40537-015-0029-9
5. http://www.sciencedirect.com/science/article/pii/S0957417414008082
6. https://hadoop.apache.org/docs/r2.8.0/hadoop-project-dist/hadoop-hdfs/HDFSHighAvailabilityWithQJM.html
7. http://jmcauley.ucsd.edu/data/amazon/links.html

8. http://hadoop.apache.org/docs/stable/hadoop-project-dist/hadoop-hdfs/HDFSHighAvailabilityWithQJM.html
9. https://hadoop.apache.org/docs/r2.7.2/hadoop-yarn/hadoop-yarn-site/YARN.html
10. https://zookeeper.apache.org/
11. https://hortonworks.com/blog/namenode-high-availability-in-hdp-2-0/
12. http://docs.aws.amazon.com/emr/latest/ManagementGuide/emr-overview-arch.html
13. https://www.linkedin.com/pulse/architecture-behind-aws-s3-native-filesystem-saugata-ghosh/
14. https://aws.amazon.com/emr/pricing/
15. https://aws.amazon.com/ec2/pricing/on-demand/
16. https://aws.amazon.com/s3/pricing/
17. https://aws.amazon.com/ebs/pricing/
18. J. McAuley, C. Targett, J. Shi, A. van den Hengel. Image-based recommendations on styles and substitutes. SIGIR, 2015