COSC 6376: Grid/Cloud Computing

Instructor: Dr. Weidong Shi (Larry), Ph.D.

Homework 1

===

K-means Clustering of Netflix Data

===

Project report

by

Venkata Yeshes Meka
PSID: 1141507
vmeka@uh.edu

Contents

1. Introduction:

K-means:

K-means clustering is one of the simplest clustering algorithms. It is called k-means because it iteratively improves our partition of the data into k sets. We choose k initial points and mark each as a center point for one of the k sets. Then for every item in the total data set we mark which of the k sets it is closest to. We then find the average center of each set, by averaging the points which are closest to the set. With the new set of centers we repeat the algorithm until the centroids converge.
The main problem with this algorithm is that it is not scalable to large sizes. As the dataset size increases the algorithm cannot perform efficiently. To circumvent this problem we need to follow a new approach to expedite the partition of dataset into reasonable subsets. This is referred to as Canopy clustering.

Canopy Clustering:

Canopy Clustering is a new technique for clustering these high dimensional datasets. The key idea involves using a cheap, approximate distance measure to efficiently divide the data into overlapping subsets called as *canopies*. Then clustering is performed by measuring exact distances only between points that occur in a common canopy. Using canopies, large clustering problems that were formerly impossible become practical. Under reasonable assumptions about the cheap distance metric, this reduction in computational cost comes without any loss in clustering accuracy. Canopies can be applied to many domains and used with a variety of clustering approaches including K-means.

2. Experimental Setup

Goal:

Cluster the Netflix movies using K-means clustering.
1. Understand the k-means and canopy clustering algorithm and their relationship.
2. Implement these algorithm in MapReduce.
3. Analyze the effect of running these algorithms on a large dataset.

Tools Used:

1. Cloudera virtual machine with Hadoop eco-system installed.
2. Eclipse.
3. Java 1.6 or higher.
4. Amazon Web Services
    a. Amazon Elastic MapReduce (Amazon EMR): for running the MapReduce jobs on Amazon Web Services.
    b. Amazon Simple Storage Service(Amazon S3): for storing the input datasets and output of each intermediate step during the clustering process.

Dataset:
The dataset is a part of the Netflix movie database having the information about the ratings given by different users to different movies.

The basic data-format of the dataset is as follows.

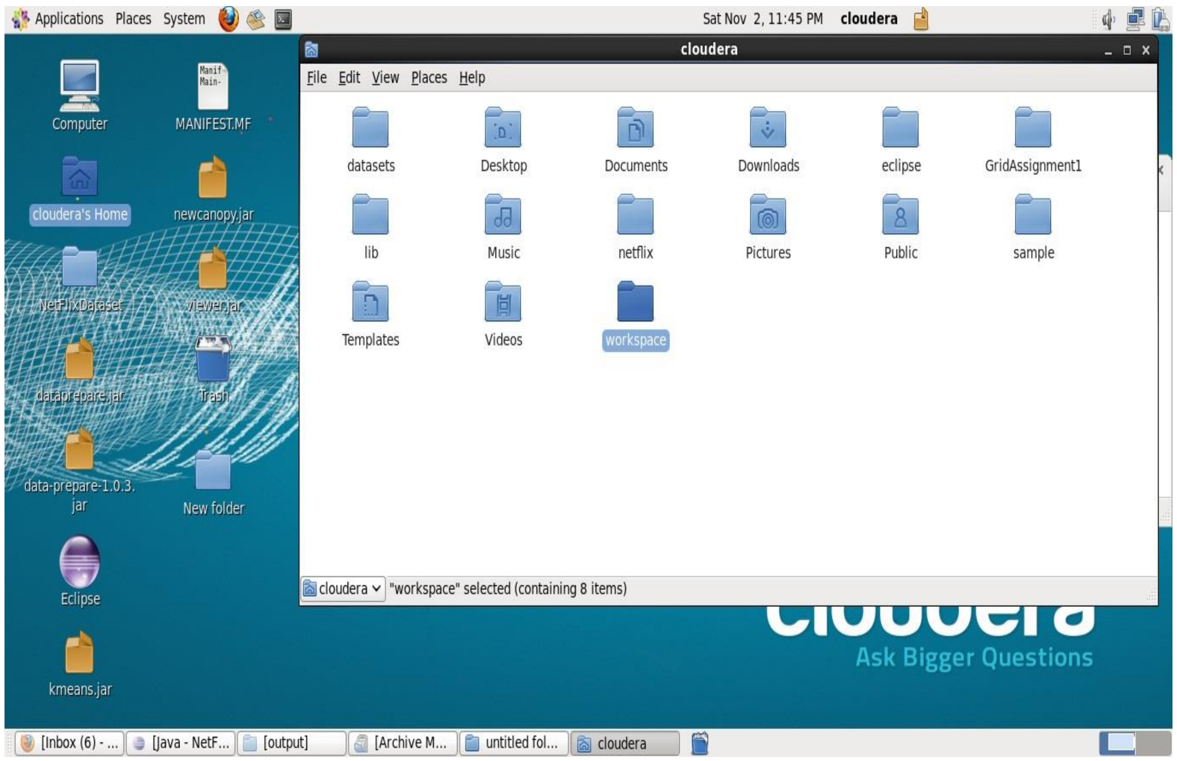
1. Each movie has a separate file with the first line of the file being the movie id.
2. From the second line of each file the data is organized as shown below.

< rater_id, rating, date >

Rater_id -> The id of the user who rates
Rating -> The rating given by the user
Date-> Date on which the user gave the rating

3. Configuring the eco-system

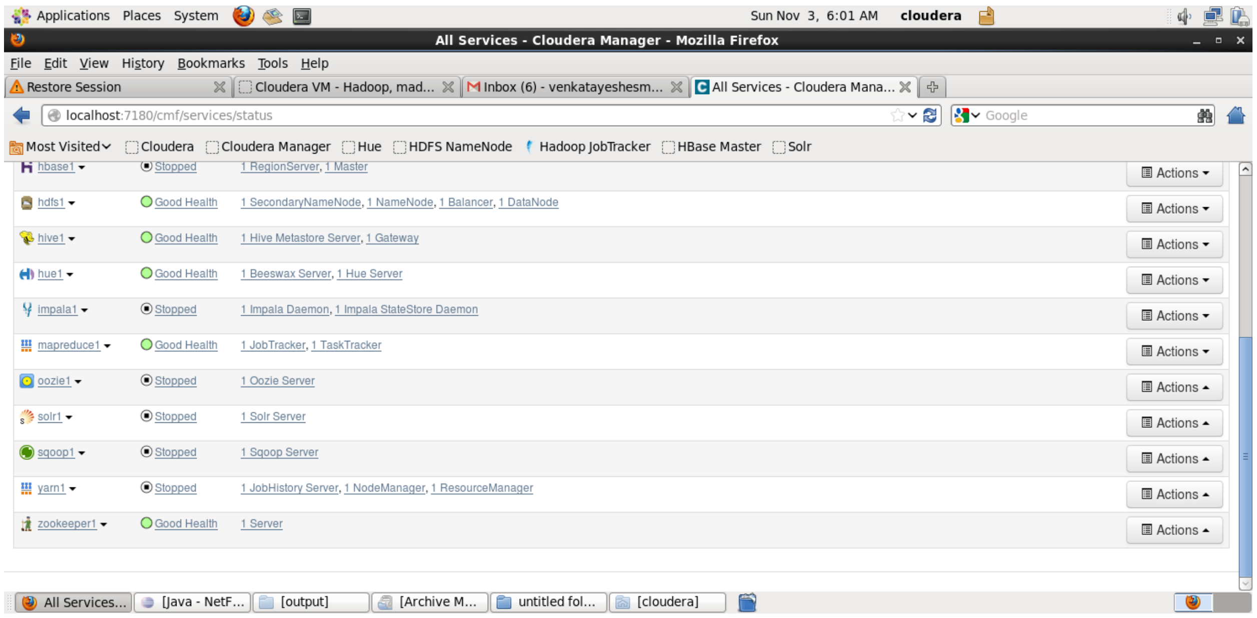
a. Configuring HDFS of Cloudera virtual machine/ to run on local machine.



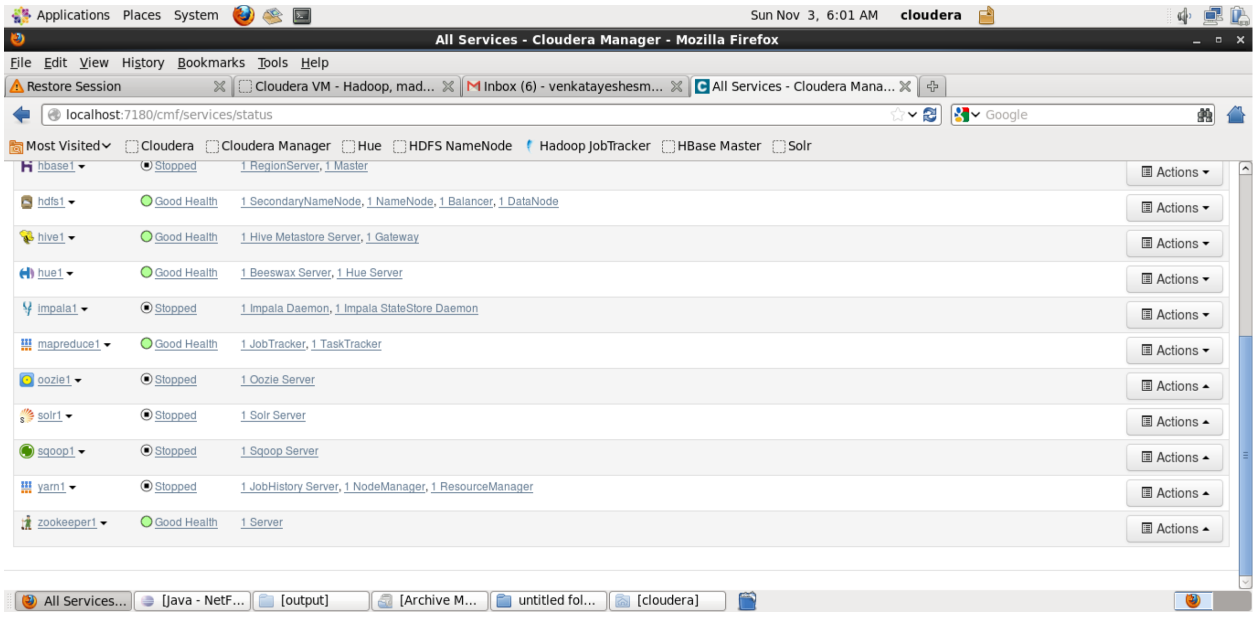
The Cloudera distribution comes with a graphical interface for the HDFS system with pre-installed Hadoop eco-system. The folder "Cloudera's-home" is the HDFS for the Hadoop eco –system installed on local machine.

The Cloudera distribution comes with a management console for the Hadoop eco-system called “Cloudera Manager”, which provides the graphical interface to control the tasks.

The jobs to cluster the Netflix movies can be run using this console. The below screenshots show the management console for the Cloudera manager. It has all the services of the Hadoop eco-system like HDFS, Hive, MapReduce, ZooKeeper etc.


Applications Places System
Sun Nov 3, 6:01 AM cloudera
All Services - Cloudera Manager - Mozilla Firefox
File Edit View History Bookmarks Tools Help
Restore Session
Cloudera VM - Hadoop, mad...
Inbox (6) - venkatayeshesm...
All Services - Cloudera Mana...
localhost:7180/cmf/services/status
Google
Most Visited Cloudera Cloudera Manager Hue HDFS NameNode Hadoop JobTracker HBase Master Solr
All Services... [Java - NetF... [output] [Archive M... untitled fol... [cloudera]

hbase1	Stopped	1 RegionServer, 1 Master	Actions
hdfs1	Good Health	1 SecondaryNameNode, 1 NameNode, 1 Balancer, 1 DataNode	Actions
hive1	Good Health	1 Hive Metastore Server, 1 Gateway	Actions
hue1	Good Health	1 Beeswax Server, 1 Hue Server	Actions
impala1	Stopped	1 Impala Daemon, 1 Impala StateStore Daemon	Actions
mapreduce1	Good Health	1 JobTracker, 1 TaskTracker	Actions
oozie1	Stopped	1 Oozie Server	Actions
solr1	Stopped	1 Solr Server	Actions
sqoop1	Stopped	1 Sqoop Server	Actions
yarn1	Stopped	1 JobHistory Server, 1 NodeManager, 1 ResourceManager	Actions
zookeeper1	Good Health	1 Server	Actions



Applications Places System
Sun Nov 3, 6:01 AM cloudera
All Services - Cloudera Manager - Mozilla Firefox
File Edit View History Bookmarks Tools Help
Restore Session
Cloudera VM - Hadoop, mad...
Inbox (6) - venkatayeshesm...
All Services - Cloudera Mana...
localhost:7180/cmf/services/status
Google
Most Visited Cloudera Cloudera Manager Hue HDFS NameNode Hadoop JobTracker HBase Master Solr
All Services... [Java - NetF... [output] [Archive M... untitled fol... [cloudera]

hbase1	Stopped	1 RegionServer, 1 Master	Actions
hdfs1	Good Health	1 SecondaryNameNode, 1 NameNode, 1 Balancer, 1 DataNode	Actions
hive1	Good Health	1 Hive Metastore Server, 1 Gateway	Actions
hue1	Good Health	1 Beeswax Server, 1 Hue Server	Actions
impala1	Stopped	1 Impala Daemon, 1 Impala StateStore Daemon	Actions
mapreduce1	Good Health	1 JobTracker, 1 TaskTracker	Actions
oozie1	Stopped	1 Oozie Server	Actions
solr1	Stopped	1 Solr Server	Actions
sqoop1	Stopped	1 Sqoop Server	Actions
yarn1	Stopped	1 JobHistory Server, 1 NodeManager, 1 ResourceManager	Actions
zookeeper1	Good Health	1 Server	Actions

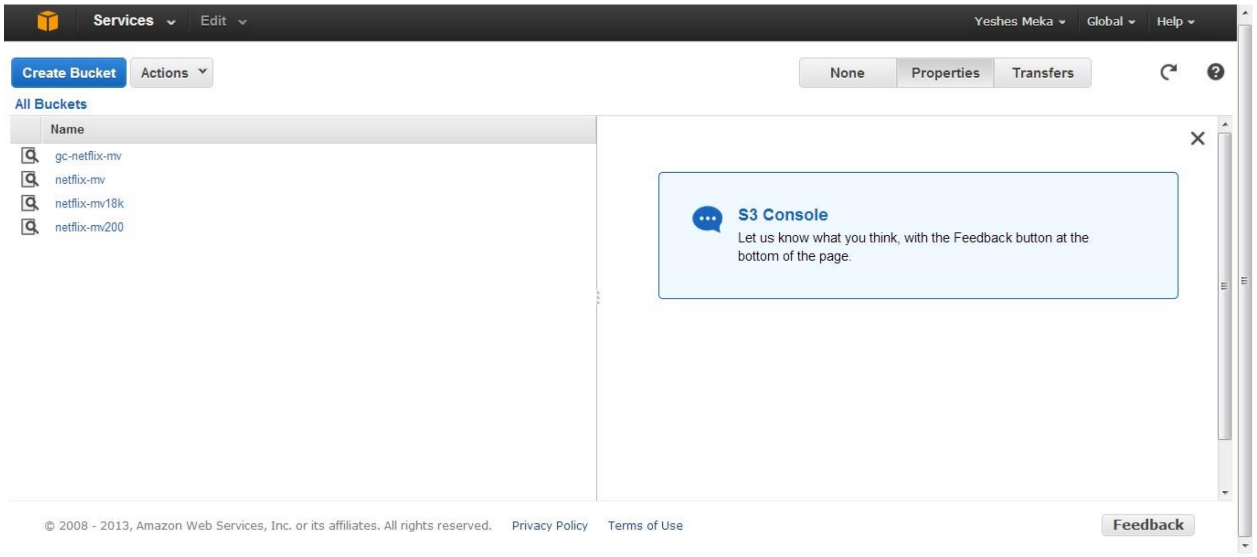
b. *Configuring Amazon Web Services (Amazon S3 & Amazon EMR)*

Amazon Simple Storage Service:

Amazon S3 provides facility to store data on cloud. This data can be used to run the MapReduce jobs using Amazon Elastic MapReduce.

Steps to upload Data on Amazon s3:

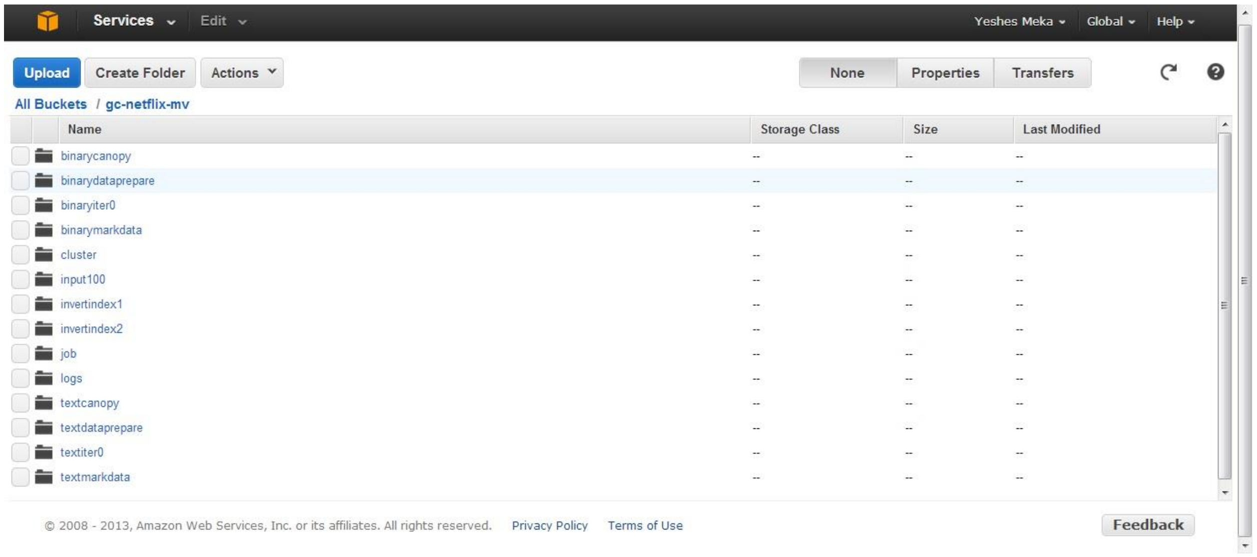
i. Login to Amazon Web Services.
ii. Create a "Bucket", which is a folder on the Amazon S3.
iii. Upload data into the Bucket.



S3 Console
Let us know what you think, with the Feedback button at the bottom of the page.

© 2008 - 2013, Amazon Web Services, Inc. or its affiliates. All rights reserved. Privacy Policy Terms of Use

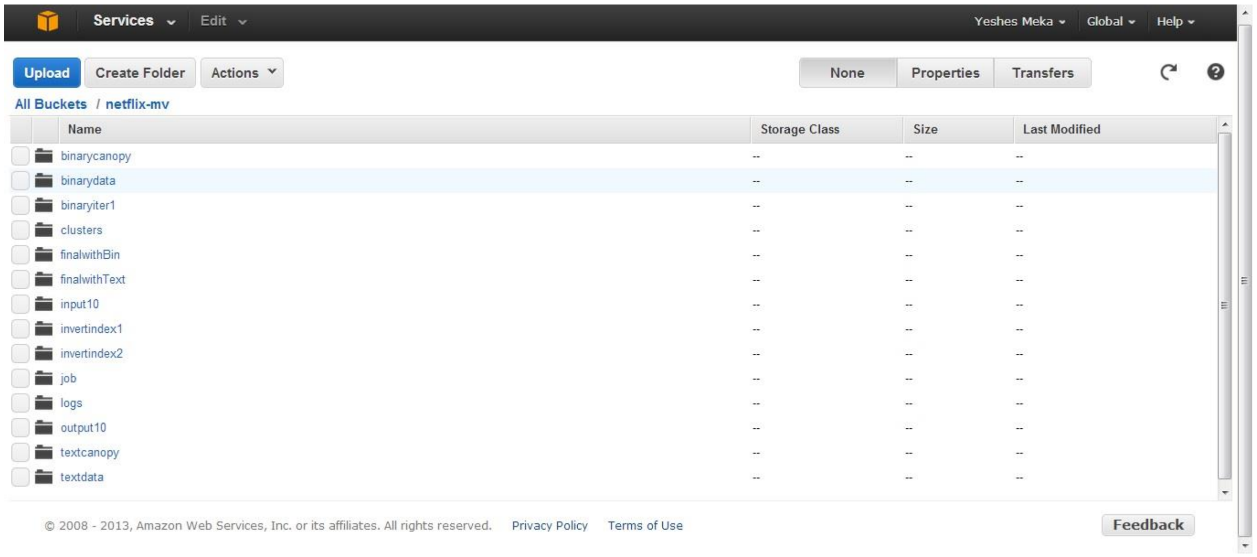
iv. Click on upload to put data onto the cloud.



Name	Storage Class	Size	Last Modified
binarycanopy	--	--	--
binarydataprepare	--	--	--
binaryiter0	--	--	--
binarymarkdata	--	--	--
cluster	--	--	--
input100	--	--	--
invertindex1	--	--	--
invertindex2	--	--	--
job	--	--	--
logs	--	--	--
textcanopy	--	--	--
textdataprepare	--	--	--
textiter0	--	--	--
textmarkdata	--	--	--

© 2008 - 2013, Amazon Web Services, Inc. or its affiliates. All rights reserved. Privacy Policy Terms of Use

The below screenshot shows the contents of a bucket after running the clustering job on Netflix dataset.

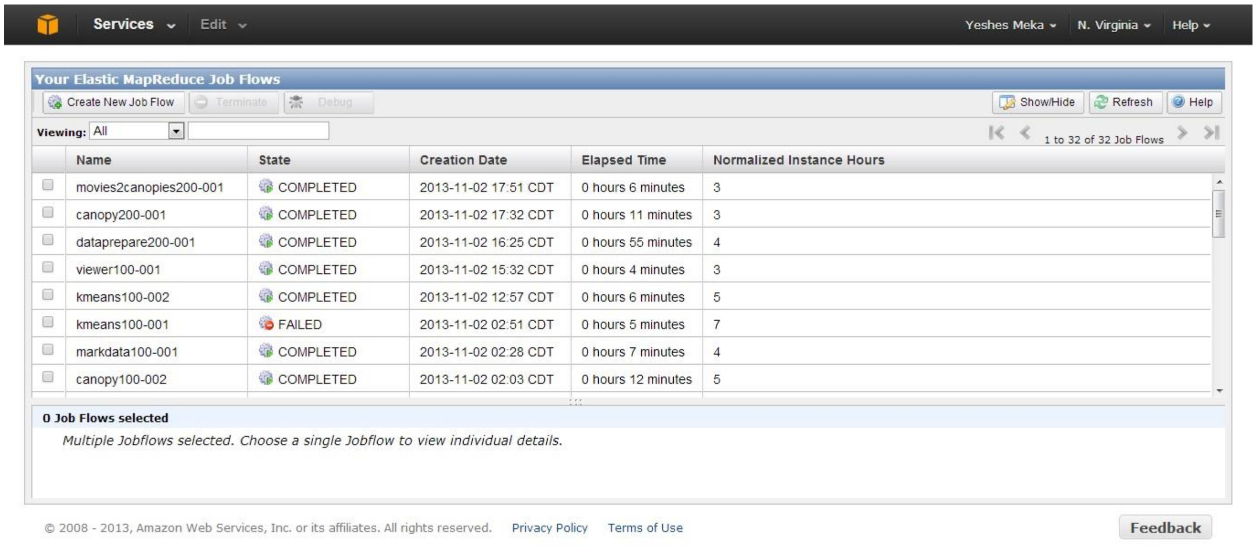


Name	Storage Class	Size	Last Modified
binarycanopy	--	--	--
binarydata	--	--	--
binaryiter1	--	--	--
clusters	--	--	--
finalwithBin	--	--	--
finalwithText	--	--	--
input10	--	--	--
invertindex1	--	--	--
invertindex2	--	--	--
job	--	--	--
logs	--	--	--
output10	--	--	--
textcanopy	--	--	--
textdata	--	--	--

© 2008 - 2013, Amazon Web Services, Inc. or its affiliates. All rights reserved. Privacy Policy Terms of Use

Amazon Elastic MapReduce:

Amazon EMR can be used to run the MapReduce jobs. It uses the Hadoop core 1.0.3 library to run the jobs. The below image shows the sample of EMR console.



Name	State	Creation Date	Elapsed Time	Normalized Instance Hours
movies2canopies200-001	COMPLETED	2013-11-02 17:51 CDT	0 hours 6 minutes	3
canopy200-001	COMPLETED	2013-11-02 17:32 CDT	0 hours 11 minutes	3
dataprepare200-001	COMPLETED	2013-11-02 16:25 CDT	0 hours 55 minutes	4
viewer100-001	COMPLETED	2013-11-02 15:32 CDT	0 hours 4 minutes	3
kmeans100-002	COMPLETED	2013-11-02 12:57 CDT	0 hours 6 minutes	5
kmeans100-001	FAILED	2013-11-02 02:51 CDT	0 hours 5 minutes	7
markdata100-001	COMPLETED	2013-11-02 02:28 CDT	0 hours 7 minutes	4
canopy100-002	COMPLETED	2013-11-02 02:03 CDT	0 hours 12 minutes	5

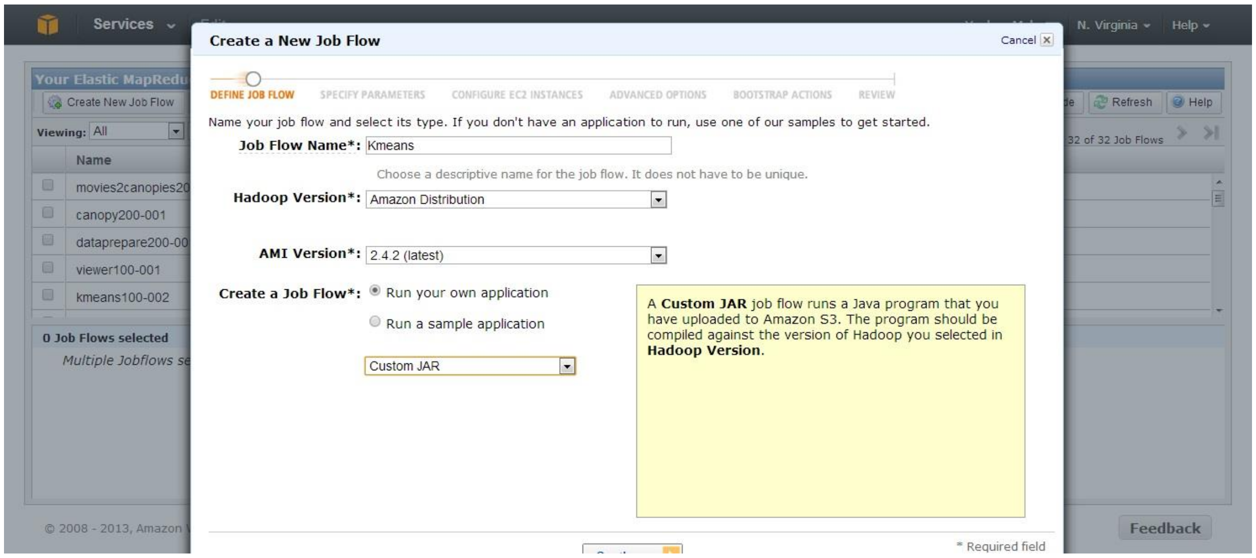
0 Job Flows selected

Multiple Jobflows selected. Choose a single Jobflow to view individual details.

© 2008 - 2013, Amazon Web Services, Inc. or its affiliates. All rights reserved. Privacy Policy Terms of Use

The below steps show how to configure and run a job.

i. Start the Amazon EMR console using the AWS Management console. Click on "create new workflow".



Create a New Job Flow

Name your job flow and select its type. If you don't have an application to run, use one of our samples to get started.

Job Flow Name*: Kmeans

Choose a descriptive name for the job flow. It does not have to be unique.

Hadoop Version*: Amazon Distribution

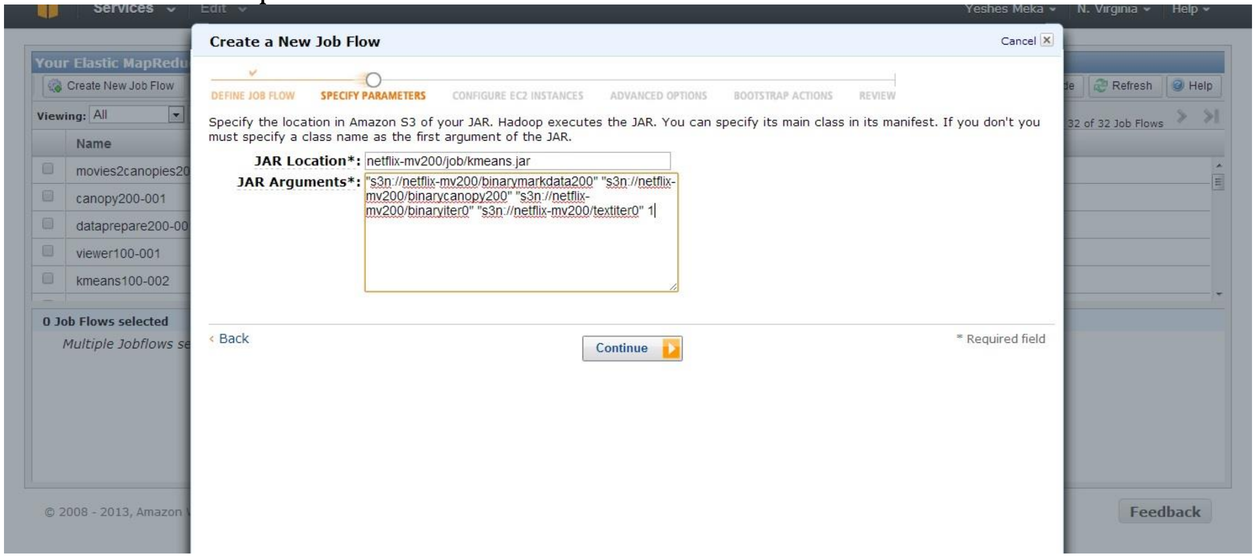
AMI Version*: 2.4.2 (latest)

Create a Job Flow*: (●) Run your own application ( ) Run a sample application

Custom JAR

A **Custom JAR** job flow runs a Java program that you have uploaded to Amazon S3. The program should be compiled against the version of Hadoop you selected in **Hadoop Version**.

ii. Enter the details of the "jar" file which has the code for the MapReduce job and set the arguments with the input, output and the requirements with respect to each step.



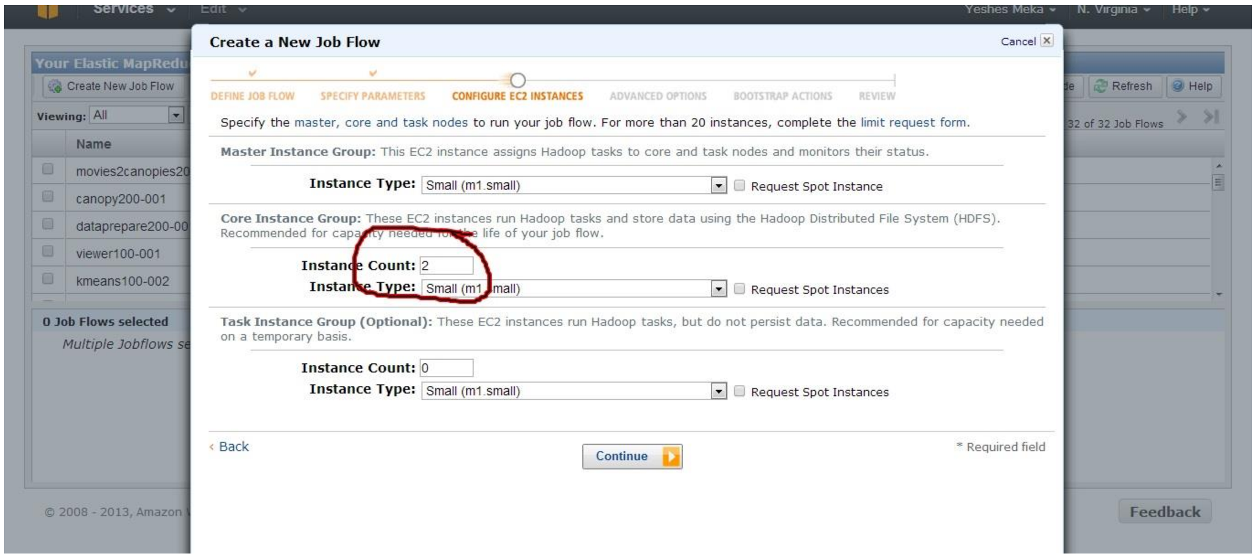
Create a New Job Flow

Specify the location in Amazon S3 of your JAR. Hadoop executes the JAR. You can specify its main class in its manifest. If you don't you must specify a class name as the first argument of the JAR.

JAR Location*: netflix-mv200/job/kmeans.jar

JAR Arguments*: "s3n://netflix-mv200/binarymarkdata200" "s3n://netflix-mv200/binarycanopy200" "s3n://netflix-mv200/binaryiter0" "s3n://netflix-mv200/textiter0" 1

iii. Select the number of instances and the types of instance required based on the nature of the task that should be run.



Create a New Job Flow

Specify the master, core and task nodes to run your job flow. For more than 20 instances, complete the limit request form.

Master Instance Group: This EC2 instance assigns Hadoop tasks to core and task nodes and monitors their status.

Instance Type: Small (m1.small) — Request Spot Instance

Core Instance Group: These EC2 instances run Hadoop tasks and store data using the Hadoop Distributed File System (HDFS). Recommended for capacity needed for the life of your job flow.

Instance Count: 2

Instance Type: Small (m1.small) — Request Spot Instances

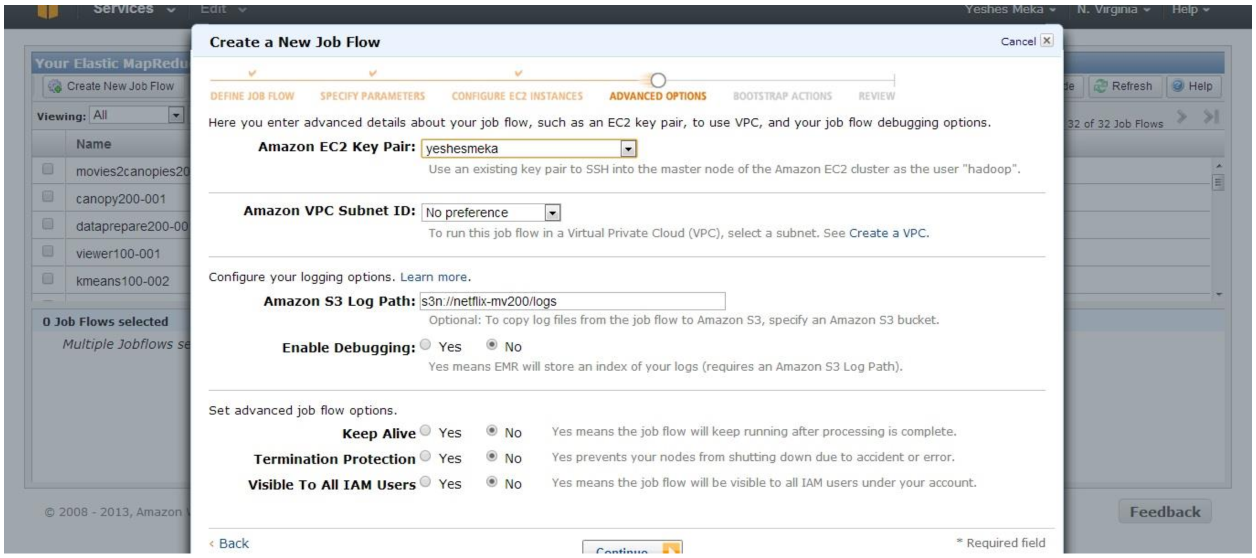
Task Instance Group (Optional): These EC2 instances run Hadoop tasks, but do not persist data. Recommended for capacity needed on a temporary basis.

Instance Count: 0

Instance Type: Small (m1.small) — Request Spot Instances

* Required field

iv. Provide the EC2 Key pair generated in the Amazon EC2.



Create a New Job Flow

Here you enter advanced details about your job flow, such as an EC2 key pair, to use VPC, and your job flow debugging options.

Amazon EC2 Key Pair: yeshesmeka
Use an existing key pair to SSH into the master node of the Amazon EC2 cluster as the user "hadoop".

Amazon VPC Subnet ID: No preference
To run this job flow in a Virtual Private Cloud (VPC), select a subnet. See Create a VPC.

Configure your logging options. Learn more.

Amazon S3 Log Path: s3n://netflix-mv200/logs
Optional: To copy log files from the job flow to Amazon S3, specify an Amazon S3 bucket.

Enable Debugging: Yes [✓ No]
Yes means EMR will store an index of your logs (requires an Amazon S3 Log Path).

Set advanced job flow options.

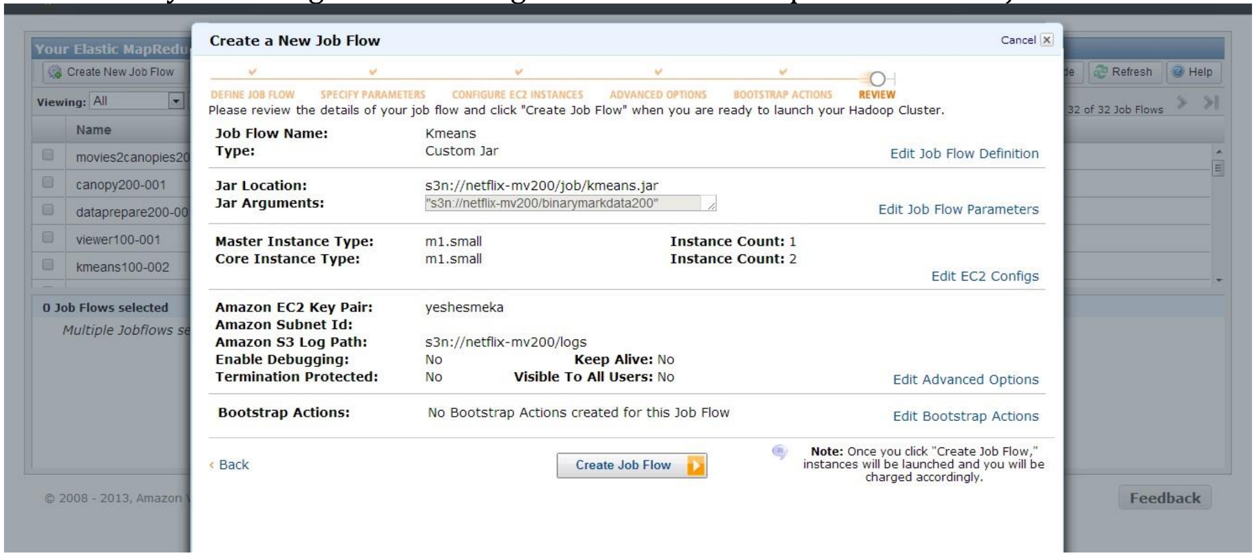
Keep Alive: Yes [✓ No] — Yes means the job flow will keep running after processing is complete.

Termination Protection: Yes [✓ No] — Yes prevents your nodes from shutting down due to accident or error.

Visible To All IAM Users: Yes [✓ No] — Yes means the job flow will be visible to all IAM users under your account.

* Required field

v. Verify the configuration settings and follow the steps to create the jobs.



Please review the details of your job flow and click "Create Job Flow" when you are ready to launch your Hadoop Cluster.

Job Flow Name: Kmeans
Type: Custom Jar

Jar Location: s3n://netflix-mv200/job/kmeans.jar
Jar Arguments: "s3n://netflix-mv200/binarymarkdata200"

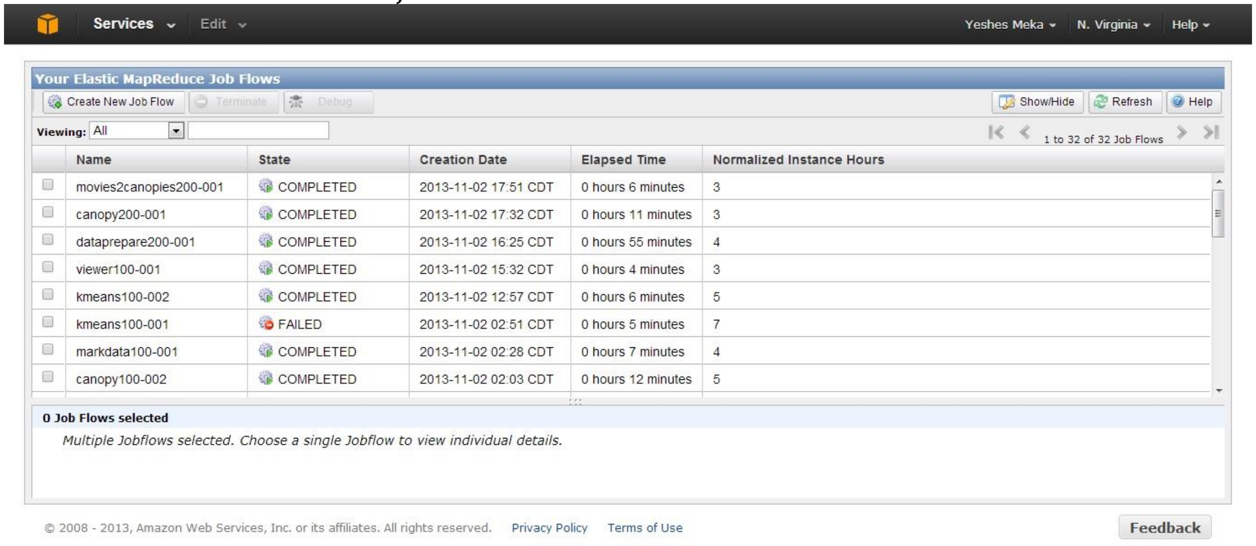
Master Instance Type: m1.small **Instance Count:** 1
Core Instance Type: m1.small **Instance Count:** 2

Amazon EC2 Key Pair: yeshesmeka
Amazon Subnet Id:
Amazon S3 Log Path: s3n://netflix-mv200/logs
Enable Debugging: No **Keep Alive:** No
Termination Protected: No **Visible To All Users:** No

Bootstrap Actions: No Bootstrap Actions created for this Job Flow

Note: Once you click "Create Job Flow," instances will be launched and you will be charged accordingly.

The below screens show the jobs and the statuses.



Name	State	Creation Date	Elapsed Time	Normalized Instance Hours
movies2canopies200-001	COMPLETED	2013-11-02 17:51 CDT	0 hours 6 minutes	3
canopy200-001	COMPLETED	2013-11-02 17:32 CDT	0 hours 11 minutes	3
dataprepare200-001	COMPLETED	2013-11-02 16:25 CDT	0 hours 55 minutes	4
viewer100-001	COMPLETED	2013-11-02 15:32 CDT	0 hours 4 minutes	3
kmeans100-002	COMPLETED	2013-11-02 12:57 CDT	0 hours 6 minutes	5
kmeans100-001	FAILED	2013-11-02 02:51 CDT	0 hours 5 minutes	7
markdata100-001	COMPLETED	2013-11-02 02:28 CDT	0 hours 7 minutes	4
canopy100-002	COMPLETED	2013-11-02 02:03 CDT	0 hours 12 minutes	5

0 Job Flows selected

Multiple Jobflows selected. Choose a single Jobflow to view individual details.

© 2008 - 2013, Amazon Web Services, Inc. or its affiliates. All rights reserved. Privacy Policy Terms of Use

4. Program Sequence

The task to cluster the given Netflix movies dataset can be performed in five stages. The output of each stage is the input to the subsequent step. The sequence of task consists of the following stages:

i. ***Data pre-processing/ preparation phase****:*

In this phase, the input dataset with all the movies in separate files are aggregated into one file with all the input files being lined up as a single line.
i.e.,
<movie id> to <userId, rating>.
The "DataPrepare" part of the code in the Code folder is the code for this phase. It takes the individual input files, runs the Mappers and Reducers to combine the files into a single file with each record in the above described format.

Data Structures used for MapReduce step: ArrayList
Input: The individual files containing Movie Id mapped to respective userID: Ratings pair.
Sample output:

```
3
     1025579:4,712664:5,1331154:4,2632461:3,44937:5,656399:4,439011:1,14
36762:3,1644750:3,2031561:4,616720:4,2467008:4,975874:5,701730:2,16143
20:4,115498:3,931626:2,699878:4,1694958:3,66414:5,2519847:5,948069:3,67
315:4,704249:4,454417:4,1995318:3,2158448:5,574843:5,714960:4,620771:2,
253876:4,1632700:4,603277:3,79160:4,1859725:4,283774:5,1983667:2,22675
07:4,1813349:4,2424721:3,1275804:4,1204327:4,2143489:1,672980:5,166100
:4,2537764:5,1650301:2,553931:4,214166:3,6689:4,109089:5,1854303:1,5250
```

ii. ***Canopy formation phase****:*

In this phase the data input which is in the combined form in the output of Data preparation phase is recursively iterated to check and assign them to a nearest canopy centers. Thus this results in the formation of canopies with the canopy center.

Canopy selection algorithm used: Each mapper receives some portion of the input movies. The mapper maintains a list in memory of canopies it has generated so far. If the current movie being mapped is within some "near" threshold of an existing canopy it has generated, then do nothing. Otherwise, the current movie is a new canopy center: emit it as an intermediate value and add it to the already-created list.
The reducer does the same thing: We take all the canopy center candidates generated by all of the mappers and then perform the same process to ensure that we do not generate two canopy centers that are basically on top of one another.

The “Canopy” part of the code in the Code folder is the code for this phase.
Data Structures used for MapReduce step: HashMap
Input: Data from Data prepare stage and number of movies in the input file.
Sample output:

```
22
    22,1,2,3,4,5,6,7,8,9,10,11,12,14,15,17,16,19,18,21,20,23,25,24,27,26,29,28,31
,30,34,35,32,33,38,39,36,37,42,43,40,41,46,47,44,45,51,50,49,48,55,54,53,52,59,
58,57,56,63,62,61,60,68,69,70,71,64,65,66,67,76,77,78,79,72,73,74,75,84,87,86,
81,80,83,82,93,92,95,94,89,88,91,90,100,99,96,97
```

iii. ***Canopy to movie mapping phase**.*

In this phase the canopies created are marked to the data that is processed in data preparation phase. Given the set of canopies, map over movie data assembled in data preparation phase. Each value should now include a set of canopies under which the movie falls. The distance metric used must be same as that used in the canopy selection phase.

The “CanopiesToMovies” part of the code in the Code folder is the code for this phase.

Data Structures used for MapReduce step: HashMap
Input: Data from Prepare stage and Canopy output
Sample output:

```
3
    1025579:4,712664:5,1331154:4,2632461:3,44937:5,656399:4,439011:1,14
36762:3,1644750:3,2031561:4,616720:4,2467008:4,975874:5,701730:2,16143
20:4,115498:3,931626:2,699878:4,1694958:3,66414:5,2519847:5,948069:3,67
315:4,704249:4,454417:4,1995318:3,2158448:5,574843:5,714960:4,620771:2,
253876:4,1632700:4,603277:3,79160:4,1859725:4,283774:5,1983667:2,22675
07:4,1813349:4,2424721:3,1275804:4,1204327:4,2143489:1,672980:5,166100,
56, 79
```

iv. ***K-means clustering phase**:*

In this phase the mapper receives a movie, its user/rating pairs, and its canopies (the output of step 3). It determines which of the current set of k-centers it is closest to (from among the k-centers in the same canopy as the movie) and emits a record containing all the movie's data and its chosen k-center. The value sent to the reducer should be keyed by the k-center (which is a movieId).

The reducer receives a k-center and all movies which are bound to that k-center. It should calculate the new position of the k-center. The giant list of movies is not updated by this step: the input from step 3 is always the only data used as input to each k-means iteration step. The purpose of this step is to move the k-centers

around. So given the list of movies bound to a current k-center, we determine which movieId is the new center of the cluster. We then emit this movie's record as the new k-center.
Each Mapper should load the current list of k-centers in its configure() phase. It should also load the list of canopies to determine which canopies each k-center falls in. For the first iteration, each canopy center can be considered a k-center. Remember to use the canopy distance metric for this. Also note that the list of k-centers is split across as many files as there were reducers in the previous round.

The K-means Distance Metric:
The set of ratings for a movie given by a set of users can be thought of as a vector [user1_score, user2_score2, … ,userN_score] (our actual representation of this vector will be sparse). If a movie has not been rated by some set of users, those scores are held at zero. To evaluate the distance between two movies, we will use the cosine vector similarity metric.

The cosine vector similarity metric is expressed as follows:

For vectors A and B each of length 'n',
similarity(A, B) = sum(A_i * B_i) / (sqrt(sum(A_i^2)) * sqrt(sum(B_i^2)))
    where the sum(...) functions retrieve all A_i or B_i for 0 <= i < n

Finding the New Center:
Finding the new center is a two step process. First, compute the term-wise average vector from all the movies mapped to a given center. Then find the movie most similar to this average vector.

When the new centers are found by averaging the vectors we found it necessary to minimize the number of features per vector, allowing them to better fit into memory for the next map. To do this we picked the top hundred thousand occurring userIDs per movie and only output the average of those user ratings

The “Kmeans” part of the code in the Code folder is the code for this phase.

Data Structures used for MapReduce step: HashMap
Input: Movies marked to Canopy centers and number of iterations.
Sample output:

```
5
    1745265:5,885013:5,1997470:5,30878:1,840543:4,2477414:5,1409354:3,5
89967:4,264160:5,2084362:4,1881496:3,240705:5,1650301:2,1815614:1,1454
357:4,624145:3,1310327:4,2012897:5,17063:5,47411:5,966474:5,1985633:4,4
7195:5,2365827:5,2451669:4,378377:3,2029786:4,2183726:4,36818:5,124746
9:5,729471:4,2371776:3,1903571:5,1422810:4,2101722:4,2187025:2,122846:
5,167540:1,1518964:3,154819:3,161395:4,2083234:5,1083500:4,119496:5,440
```

```
5,1,2,3,4,6,7,8,9,10,11,12,13,14,15,17,16,19,18,21,20,23,22,25,24,27,26,29,28,31,
30,34,35,32,33,38,39,36,37,42,43,40,41,46,47,44,45,51,50,49,48,55,54,53,52,59,
58,57,56,63,62,61,60,68,69,70,71,64,65,66,67,76,77,78,79,72,73,74,75,85,84,87,
86,81,80,83,82,93,92,95,94,89,88,91,90,100,99,96,97
```

Challenges experienced during this phase: The difficulty is to determine the cosine similarity metric. When I tried with Euclid distance, the output was appropriate but the computation of cosine similarity was the major concern, given the large dataset.

v. ***Data cleanup phase****:*

In this phase the output received from K-means phase is cleaned up by removing the userId:rating maps and just having the clusters to which each movie belongs.
The output contains list with each k-center as a key, with the value being the list of movieIds attached to that center (use the mapper from step 4 to bind movieIds to a k-center).

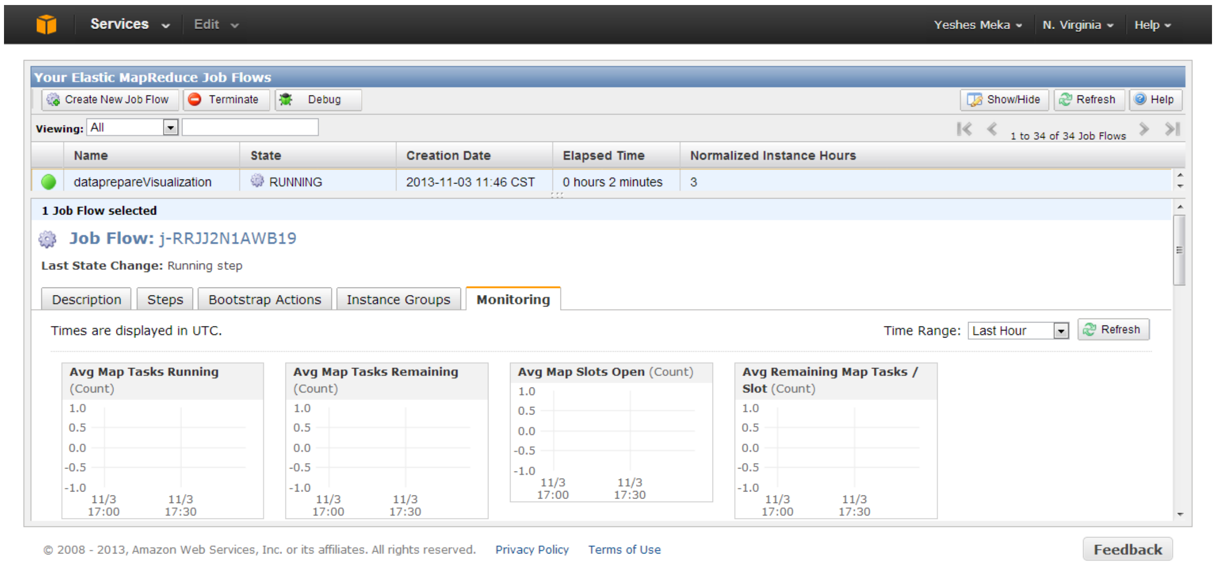
The "DataCleanup" part of the code in the Code folder is the code for this phase.

Data Structures used for MapReduce step: HashMap
Input: Output from K-means collection.
Sample output:

```
6
6,1,2,3,4,5,7,8,9,10,11,12,13,14,15,17,16,19,18,21,20,23,22,25,24,27,26,29,28,31,
30,34,35,32,33,38,39,36,37,42,43,40,41,46,47,44,45,51,50,49,48,55,54,53,52,59,
58,57,56,63,62,61,60,68,69,70,71,64,65,66,67,76,77,78,79,72,73,74,75,85,84,87,
86,81,80,83,82,93,92,95,94,89,88,91,90,100,98,99,96,97
```

5. Amazon Web Service computation visualization

The following plots show the various operations performed by the Elastic MapReduce tasks during the runtime of a job. The below plot is an example of the runtime visualizations during the data preparation step.



Viewing: All

1 to 34 of 34 Job Flows

Name	State	Creation Date	Elapsed Time	Normalized Instance Hours
datapreparеVisualization	RUNNING	2013-11-03 11:46 CST	0 hours 2 minutes	3

1 Job Flow selected

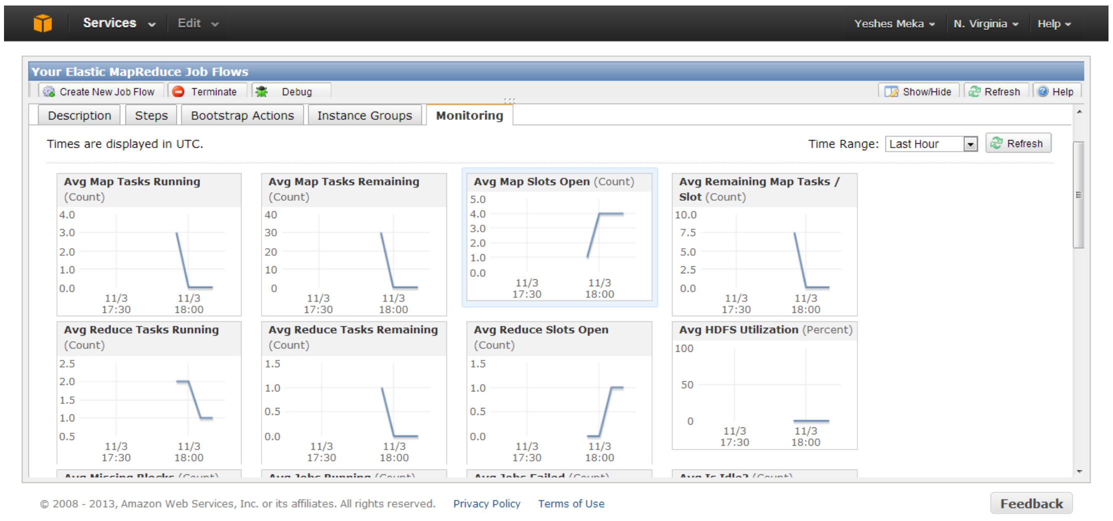
Job Flow: j-RRJJ2N1AWB19

Last State Change: Running step

Times are displayed in UTC.

Time Range: Last Hour

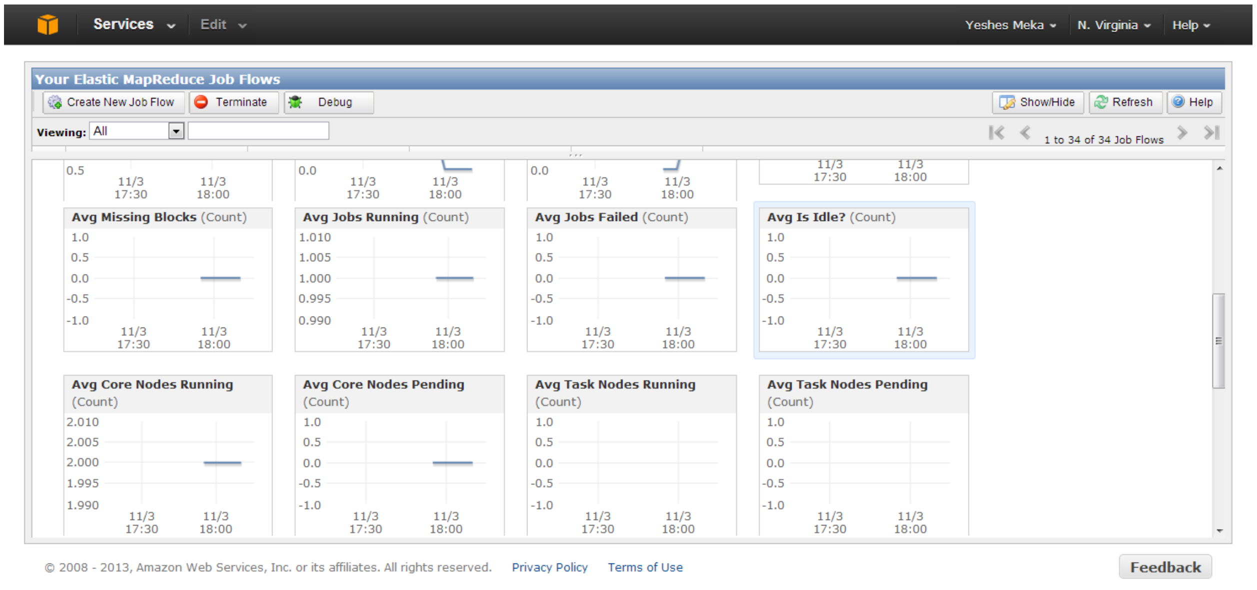
© 2008 - 2013, Amazon Web Services, Inc. or its affiliates. All rights reserved. Privacy Policy Terms of Use

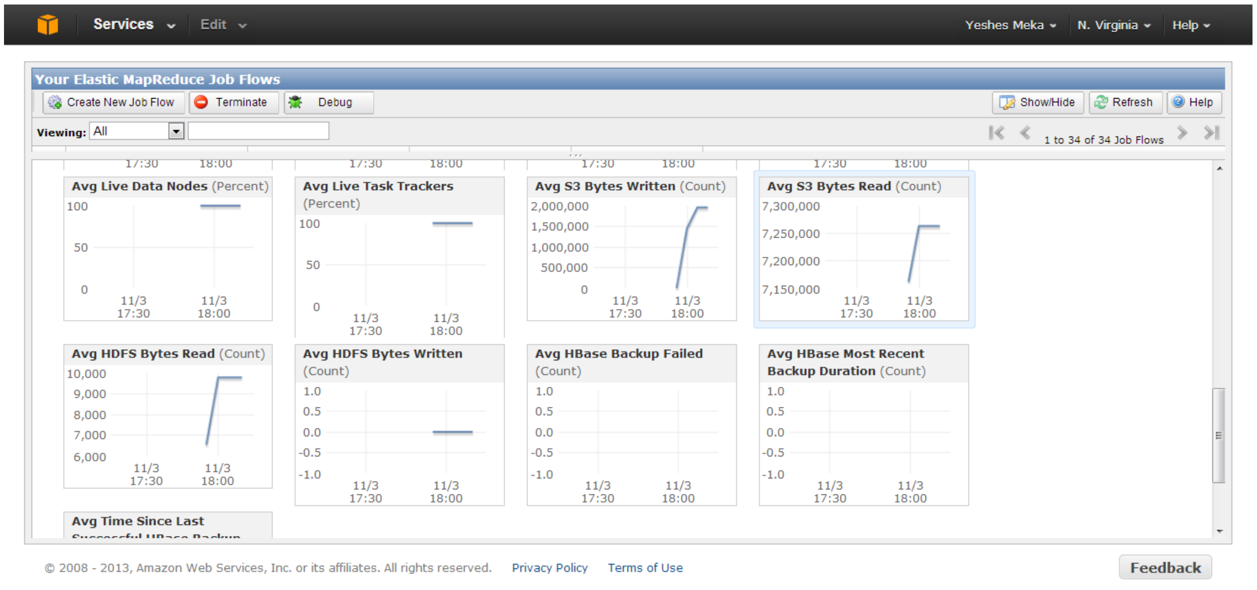

Services
Edit
Yeshes Meka
N. Virginia
Help
Your Elastic MapReduce Job Flows
Create New Job Flow
Terminate
Debug
Show/Hide
Refresh
Help
Description
Steps
Bootstrap Actions
Instance Groups
Monitoring
Avg Map Tasks Running (Count)
Avg Map Tasks Remaining (Count)
Avg Map Slots Open (Count)
Avg Remaining Map Tasks / Slot (Count)
Avg Reduce Tasks Running (Count)
Avg Reduce Tasks Remaining (Count)
Avg Reduce Slots Open (Count)
Avg HDFS Utilization (Percent)
11/3 17:30
11/3 18:00
Feedback

Times are displayed in UTC.

Time Range: Last Hour

© 2008 - 2013, Amazon Web Services, Inc. or its affiliates. All rights reserved. Privacy Policy Terms of Use


Services
Edit
Yeshes Meka
N. Virginia
Help
Your Elastic MapReduce Job Flows
Create New Job Flow
Terminate
Debug
Show/Hide
Refresh
Help
Viewing: All
1 to 34 of 34 Job Flows
Avg Missing Blocks (Count)
Avg Jobs Running (Count)
Avg Jobs Failed (Count)
Avg Is Idle? (Count)
Avg Core Nodes Running (Count)
Avg Core Nodes Pending (Count)
Avg Task Nodes Running (Count)
Avg Task Nodes Pending (Count)
11/3 17:30
11/3 18:00
© 2008 - 2013, Amazon Web Services, Inc. or its affiliates. All rights reserved.
Privacy Policy
Terms of Use
Feedback


Services
Edit
Yeshes Meka
N. Virginia
Help
Your Elastic MapReduce Job Flows
Create New Job Flow
Terminate
Debug
Show/Hide
Refresh
Help
Viewing: All
1 to 34 of 34 Job Flows
Avg Live Data Nodes (Percent)
Avg Live Task Trackers (Percent)
Avg S3 Bytes Written (Count)
Avg S3 Bytes Read (Count)
Avg HDFS Bytes Read (Count)
Avg HDFS Bytes Written (Count)
Avg HBase Backup Failed (Count)
Avg HBase Most Recent Backup Duration (Count)
Avg Time Since Last
11/3 17:30
11/3 18:00
© 2008 - 2013, Amazon Web Services, Inc. or its affiliates. All rights reserved.
Privacy Policy
Terms of Use
Feedback

6. Comparision of Running times of MapReduce Jobs

For Dataset sample size 10 Movies

SL. No	Task	No. of Instances	Runtime for task
1	Data Prepare	3	6 mins
2	Canopy	3	10 mins
3	Assign Movies to Canopy	3	7 mins
4	K-means Clustering	3	10 mins
5	Data Cleanup	3	5 mins

For Dataset sample size 100 Movies

SL. No	Task	No. of Instances	Runtime for task
1	Data Prepare	4	50 mins
2	Canopy	4	12 mins
3	Assign Movies to Canopy	4	7 mins
4	K-means Clustering	4	6 mins
5	Data Cleanup	4	4 mins

For Dataset sample size 200 Movies

SL. No	Task	No. of Instances	Runtime for task
1	Data Prepare	4	55 mins
2	Canopy	4	11 mins
3	Assign Movies to Canopy	4	13 mins
4	K-means Clustering	4	15 mins
5	Data Cleanup	4	7 mins

For Dataset sample size 1000 Movies

SL. No	Task	No. of Instances	Runtime for task
1	Data Prepare	10	4hrs 23mins
2	Canopy	10	3hrs 10 mins
3	Assign Movies to Canopy	10	55 mins
4	K-means Clustering	10	1hr 20 mins
5	Data Cleanup	10	30 mins

For Dataset sample size 17770 Movies

SL. No	Task	No. of Instances	Runtime for task
1	Data Prepare	15	5 hrs 43 mins

After analyzing the processing times for the above samples, I observed that the running time for the Data preparation Step and Canopy discovering steps took comparatively large amount of time than the subsequent steps. The inference is that since, combining the large number of files takes maximum time as each file has to be read and then processed.
Also, another observation is that with the increase in number of instances of processing, the time doesn't seem to vary any different. With large datasets, the number of instances increased the runtime could be almost linearly scalable. Further, research has to be done in order to practically prove this.

**For the complete dataset, the first phase takes 5 hrs 43 mins. The second phase ran for about 10 hrs and failed because there was restriction on the free tier and the number of instances. Hence I provided the timings for just the first phase for 17770 movies dataset.

7. Challenges

The challenges I faced during the process of running and processing the dataset are :

i. Computational complexity

Generally, the problem of working with large datasets is the computational complexity. Since the given dataset of Netflix movies consists of around 2 GB of data and with 17000 movies, the algorithms have to be optimized in order to achieve the analytic results.

Strategy: In order to understand the pattern I ran the jobs on 10 movies, 100 movies, 200 movies, 1000 movies and then tried to run for 18000 movies.

ii. Processing power

The major concern is the processing power since these BigData tasks demand high processing power to process the data. This means the availability of clusters which can be specifically used to run Hadoop MapReduce jobs.

Strategy: Initially ran the jobs on random sample of datasets in the virtual machine provided by Cloudera. Then after understanding the time taken to run and the demand on resources on my local machine I estimated how much would be required to run the entire dataset. So the work around is clustering on maximum of 1000 movies.

iii. Resources

The other major concern is the resources, since my local machine would be constrained on the resources. In order to efficiently process the entire dataset the demand of resources is more. Also, the Amazon S3 has a constraint of 5 GB data storage on the free tier, hence running the jobs for 17770 movies is a really difficult task.

Strategy: Running the jobs on Amazon Web Services free tier also has constraints on it but have much better computational power. So I ran the dataset sample of 1000 movies on the Amazon Web Services using various configurations.

iv. Time constraints

The other constraint is determining the number of instances to run the jobs on. Since the services on Amazon are on-demand and time based, it becomes imperative to estimate the running times of each job and trying to optimize the tasks.

8. *References*

[1] www.code.google.com
[2] http://cs.boisestate.edu/~amit/research/makho_ngazimbi_project.pdf
[3] http://www.drdobbs.com/database/hadoop-writing-and-running-your-first-pr/240153197
[4] http://horicky.blogspot.com/2011/04/k-means-clustering-in-map-reduce.html
[5] http://aws.amazon.com/documentation/s3/
[6] http://aws.amazon.com/documentation/emr/