CS 542 Project Report

Designing for Scale: A Simplified Facebook Framework Using Hadoop Hive

By

Zhongyuan Fu

Rob Proulx

Yupu Song

Table of Contents

1 Introduction

The preceding research project performed by our group examined Facebook's core data functions in detail. Figure 1 shows an overall map of the database solutions employed.

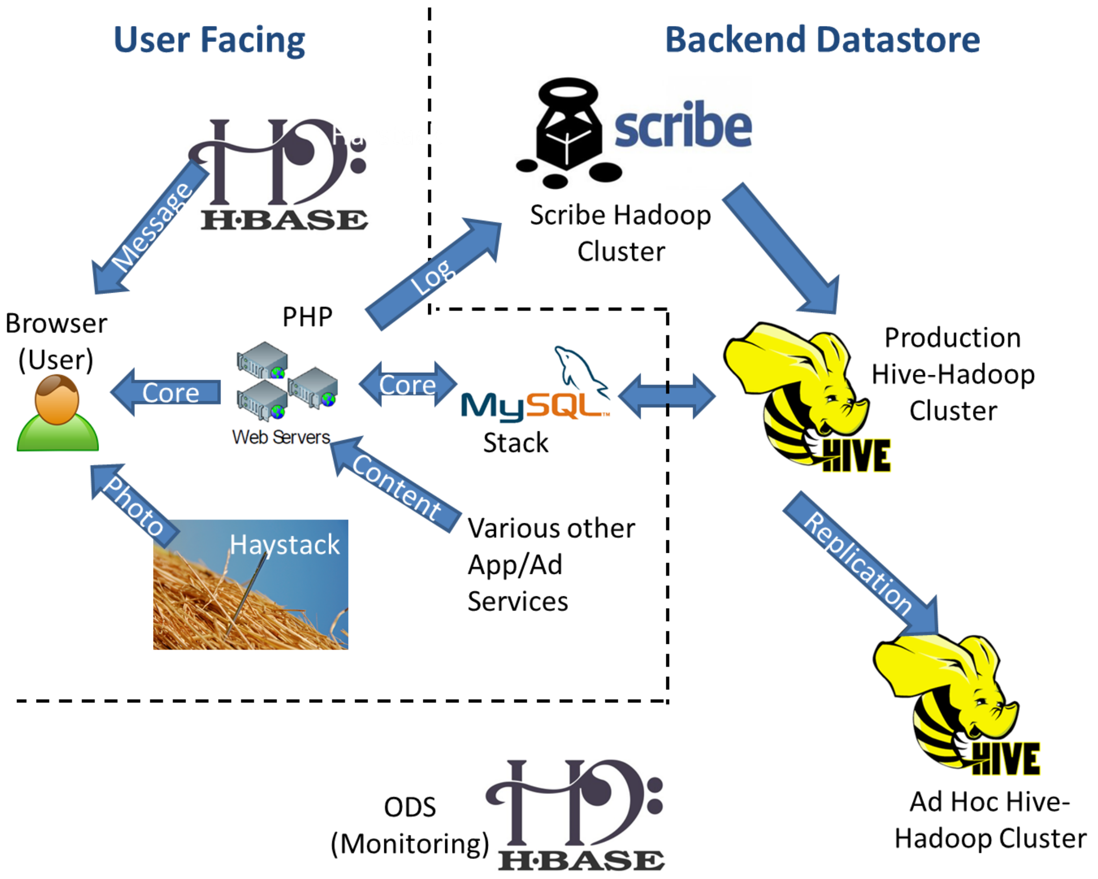


Figure 1—Facebook Database Map

Additionally, the report examined how the core functions originally were hosted on a standard LAMP (Linux, Apache HTTP Server, MySQL database, PHP web) setup. Despite incredible growth Facebook still uses a MySQL stack with a comprehensive sharding/replication and cache scheme. Facebook defends this decision citing that MySQL is quite fast, etc., but one wonders how much the fact that they presently use MySQL is a factor.

2 Project Concept

2.1 Scope

This project considers the notion of how a small company could start with a data solution capable of growing into something truly webscale, without the manual replication and queue layers.

To that end this project comprises a front-end routine that interfaces with the Hadoop backend, simulating simplified Facebook functionality. The solution also comprises bot functions to build simulated Facebook relation data.

Beyond the components that the team developed, Amazon Web Services were used to provide the backend functionality. The Amazon S3 serves as the datastore, while the EMR service (Elastic Map Reduce) servers provide the computational backend.

In the backend, the Hive engine is used to manipulate the stored S3 data. Built on Hadoop, Hive is known to be massively scalable, a truly web-scale solution.

2.2 Architecture

Figure 2 shows a plot of the overall architecture of our system. A Java based bot was used to create test data. This text-based data was then pushed manually to the S3 datastore to start the process.

The Python client, simulating facebook query functions interfaces to the S3 and EMR by SSH protocol.

The EMR can access the S3 datastore natively, both components existing on the same Amazon region.

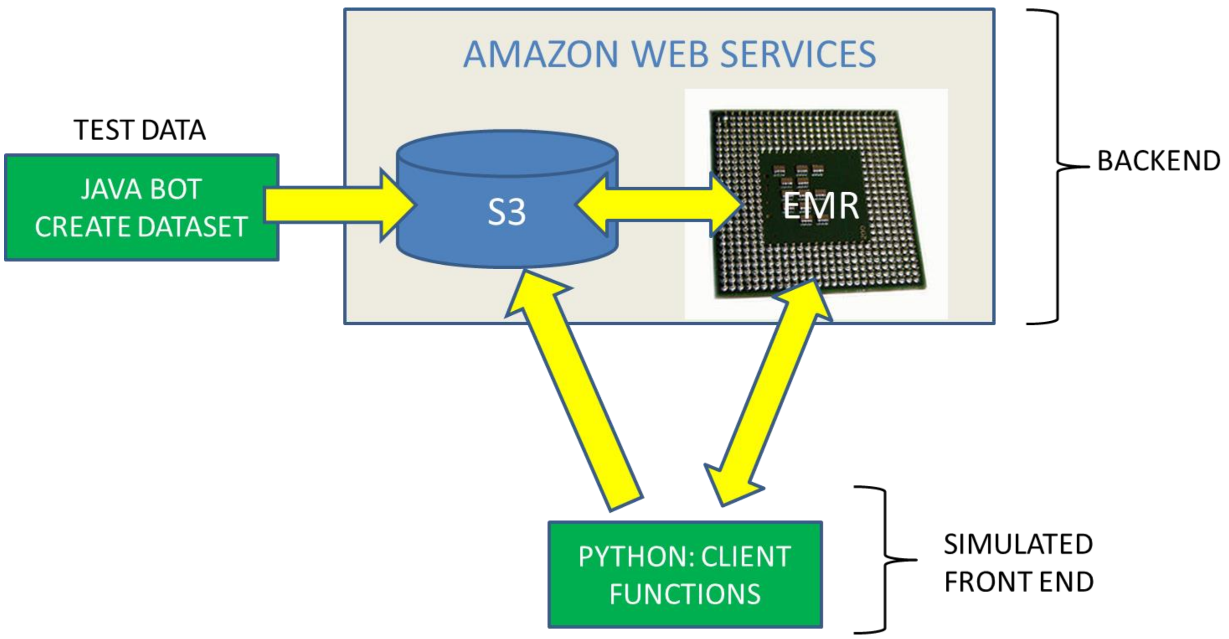


Figure 2—Architecture of Solution

2.2.1 Bot to Create Data

Initial efforts to pull actual facebook data for our test efforts were thwarted by license issues. The team investigated using the Facebook API to collect user likes and other data, but the license states that user data is not allowed to be used outside of Facebook without each user's permission. Since getting this permission would be prohibitive, another approach was taken.

A set of Java based functions to generate text-based data for the relations was created to start testing.

2.2.2 Client Functions

A series of client-side functions was written in Python to simulate how the Facebook web servers could initiate manage the Hive and S3. The boto functions[1] are used to connect to the Amazon Web Services by SSH protocol and special management of Key Pairs

2.2.3 Amazon Backend

Amazon S3 buckets provide storage for the project. Initially the bot generated files are placed into appropriately named S3 buckets.

Then an Elastic Map Reduce cluster is spawned, running Hive and Hadoop. The Client functions can connect to the backend to run the various queries and operations.

3 Implementation

In the following section, the details of the project work are presented. First, investigation of the database schema is shown. The java functions created to generate the test data are presented. Step-by-step instructions for using the Amazon Web Services including pushing data to the S3 datastore and starting an EMR cluster are shown. Finally, the python client end functions are presented, showing example query results relevant to the Facebook user experience.

3.1 Create Relation Test Data

This project created an appropriate data schema based on both the real Facebook personal pages and the test functionality targeted to prove our concept. Constraints on the data will be made in light of practical meaning and feasibility. Ideally the project would use data as close as to the real-world data as possible. However, limitations regarding the memory size and calculation speed of our test servers, have driven simplifications to our schema.

3.1.1 Facebook Schema Analysis

Facebook processes more than 500 terabytes every day[2], including 2.5 billion content items shared, 2.7 billion likes, 300 million photos uploaded, etc. How does Facebook stores such staggering amounts of data as well as use them in an efficiently way? Intricate problem as it is, a good access point is to explore

[1] https://pypi.python.org/pypi/boto

[2] http://www.cnet.com/news/facebook-processes-more-than-500-tb-of-data-daily/

the data scheme, which is also the first step for simulating users' data. Here is a relatively detailed table schema[3] for Facebook data:

[3] http://www.socialmeteor.com/wp-content/uploads/2009/05/

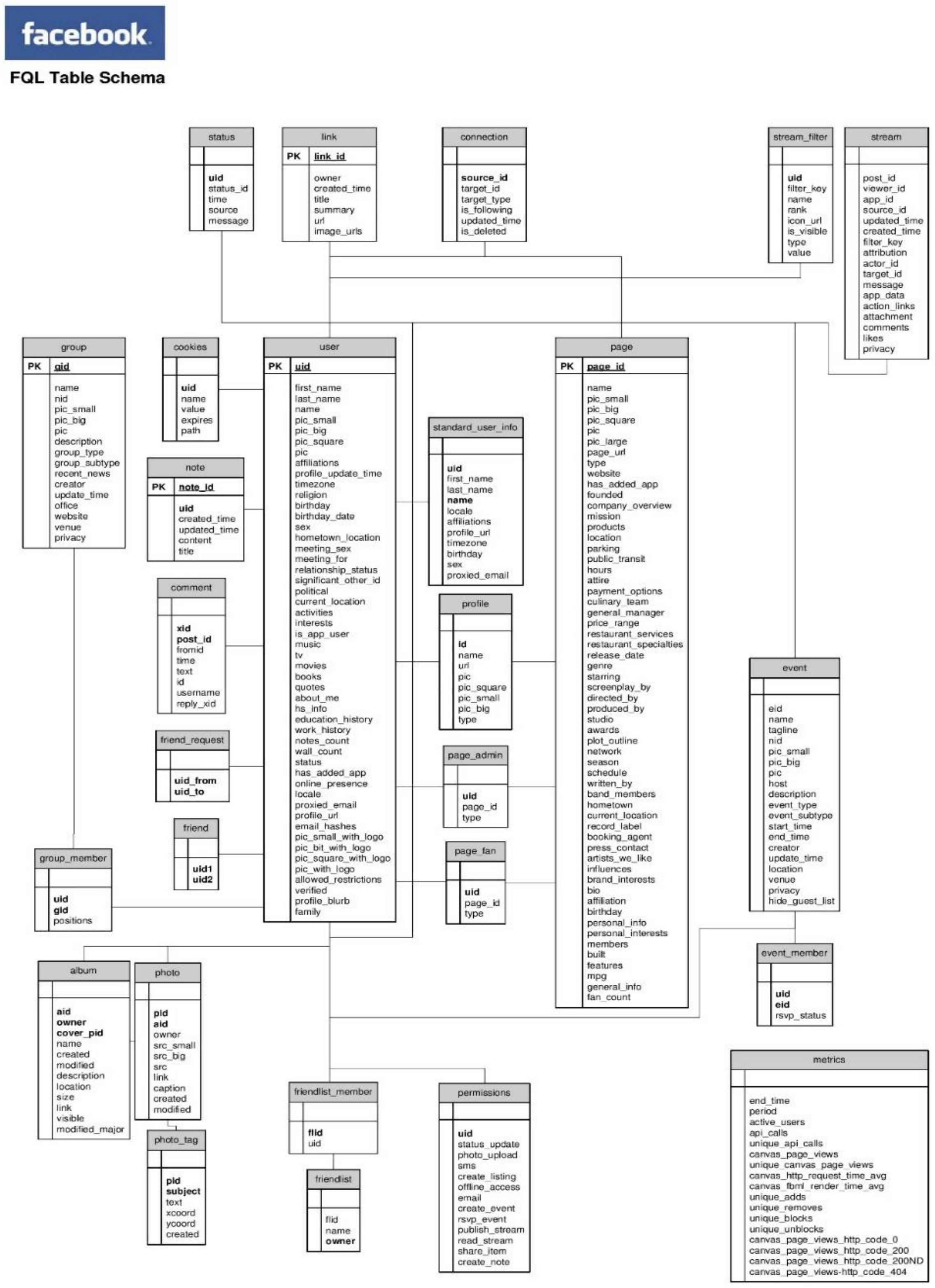
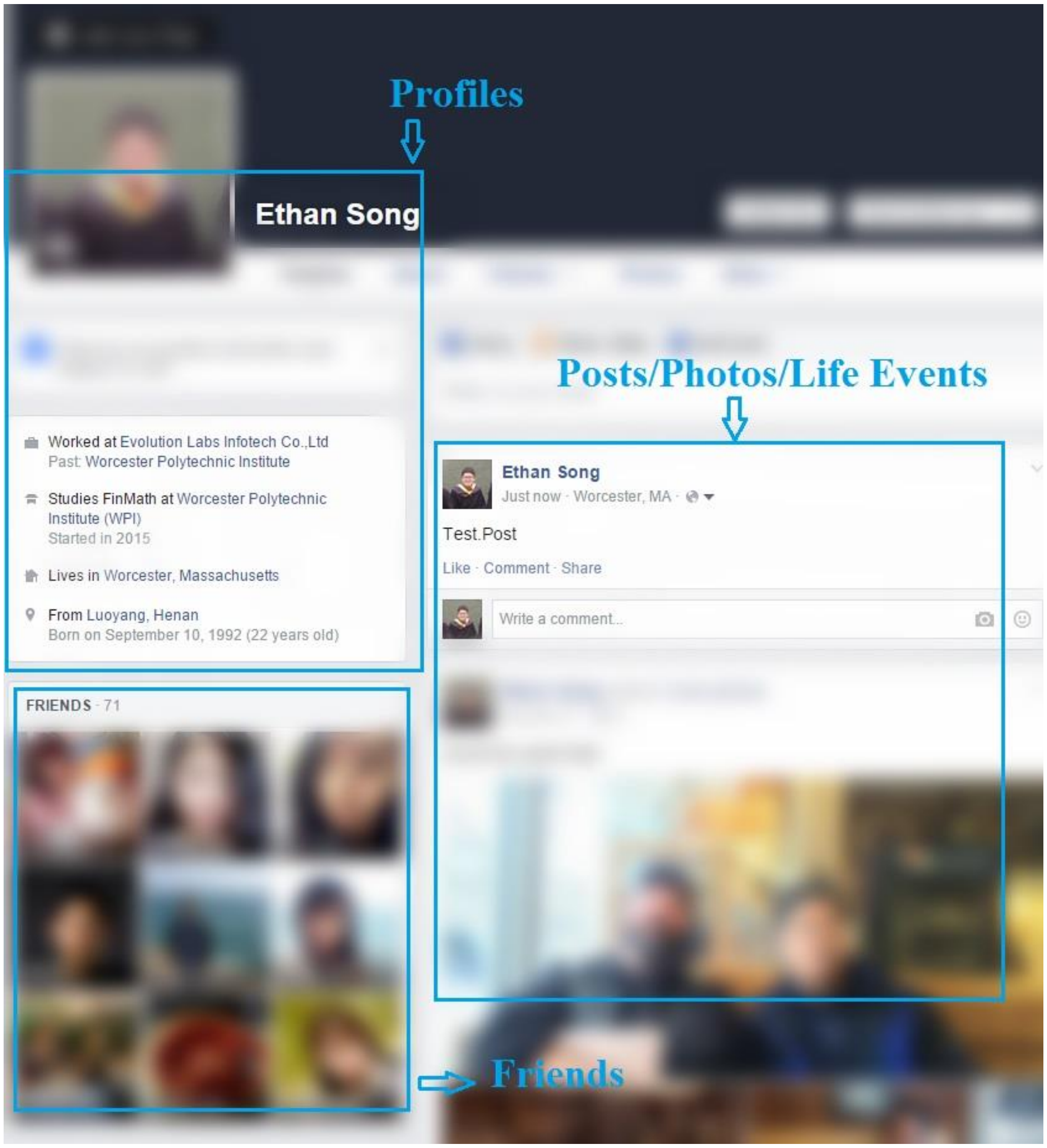


Figure 3—Facebook Schema

As can be readily seen from Figure 3, the real data schema is too complicated for our project. To that end, we have built a schema in the light of the real Facebook user's personal page and the queries we want to make.

To learn the functions of Facebook, we take inspiration from the Facebook personal page. A typical personal page is shown in Figure 4.



Worked at Evolution Labs Infotech Co.,Ltd
Past: Worcester Polytechnic Institute

Studies FinMath at Worcester Polytechnic Institute (WPI)
Started in 2015

Lives in Worcester, Massachusetts

From Luoyang, Henan
Born on September 10, 1992 (22 years old)

Ethan Song
Just now · Worcester, MA

Test Post

Figure 4—Facebook Functions

As observed, this personal page mainly consists of three parts:
1. Profiles. Including name, company, school, current location, hometown, birthdate, etc.
2. Friends list.
3. Posts, photos, life events, etc. For a post, it can be liked, shared, and commented.

Therefore, a simulated database solution should touch upon these three parts. Taking this one step further, example query functions that relevant to these three core parts are:

- Show the list of friends for a user
- Show the list of posts from a user
- Show everyone who likes a post
- Find users located in the same city
- Find posts that contain a certain word

The simplified schema developed next has been designed to handle these functions.

3.1.2 Database Design

Figure 5, below, shows the schema developed to enable the query questions above.

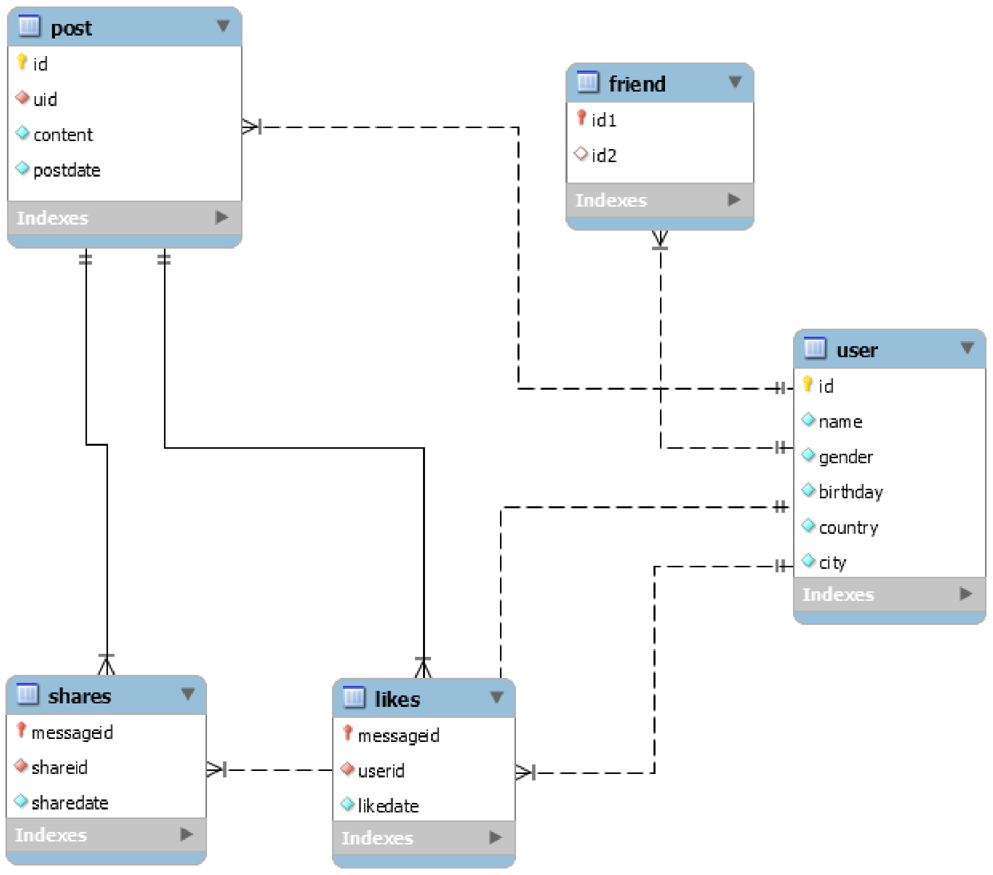


Figure 5—Simplified Schema

This database is composed of five tables, which are respectively:

1. user: id, name, gender, birthday, country, city

Note: we generate and store user's id by using integers instead of strings for simplification. The names of users derive from a names dictionary text file, and the location names derive from a system data file of Tecent QQ (an instant messaging software) which stores names of countries and cities.

Attribute	Datatype	Description	Constraint/Source
id	INT	user's unique id, simplified by using integers instead of strings	
name	STRING	user's name, including first and last name	imported from the names dictionary
gender	STRING	male or female	value='M' or 'F'
birthday	STRING	birthdate of the user	value='YYYY-MM-DD'
country	STRING	user's location, country name	imported from the location dictionary text file
city	STRING	user's location, city name	imported from the location dictionary text file

2. post: id, uid, content, postdate

Note: a user can write many posts and each post has its unique post id. The content of a post is generated by randomly picking 1 to 7 words in a vocabulary text file.

Attribute	Datatype	Description	Constraint/Source
<u>id</u>	INT	message's unique id, simplified by using integers	Not null
uid	INT	the user's id who post this message	references user.id
content	STRING	content of the post	imported from the vocabulary text file, randomly take 1 to 7 words
postdate	TIMESTAMP	timestamp of post	value='YYYY-MM-DD HH:MM:SS'

3. friend: id1, id2

Attribute	Datatype	Description	Constraint/Source
<u>id1</u>	INT	user's id	not null, references user.id
id2	INT	id of the user's friend	not null, references user.id

4. likes: messageid, userid, likedate

Attribute	Datatype	Description	Constraint/Source
<u>messageid</u>	INT	message's id	not null, references post.id
userid	INT	id of the user who likes this post	not null, references user.id
likedate	TIMESTAMP	timestamp of the like operation	not null, value='YYYY-MM-DD HH:MM:SS'

5. shares:

Attribute	Datatype	Description	Constraint/Source
<u>messageid</u>	INT	message's id	not null, references post.id
userid	INT	id of the user who shares this post	not null, references user.id
sharedate	TIMESTAMP	timestamp of the share operation	not null, value='YYYY-MM-DD HH:MM:SS'

3.1.3 Java Bot Implementation

Several Java functions were used to generate the data. This code can be found on github here https://github.com/robproulx/CS542/tree/master/src/com.

MainPicker.Java: Set the overall global sizes for the random generation (number of users, number of friends, etc.) 100 users by default.

FriendRelation.Java: Generate the friend relationships between users (100K friend pairs)

NamePicker.Java: Generate the user table (100K users)

PostPicker.Java: Generate the posts table (250K posts)

ShareAndLike.Java: Generate the like and share table. (250K total likes)

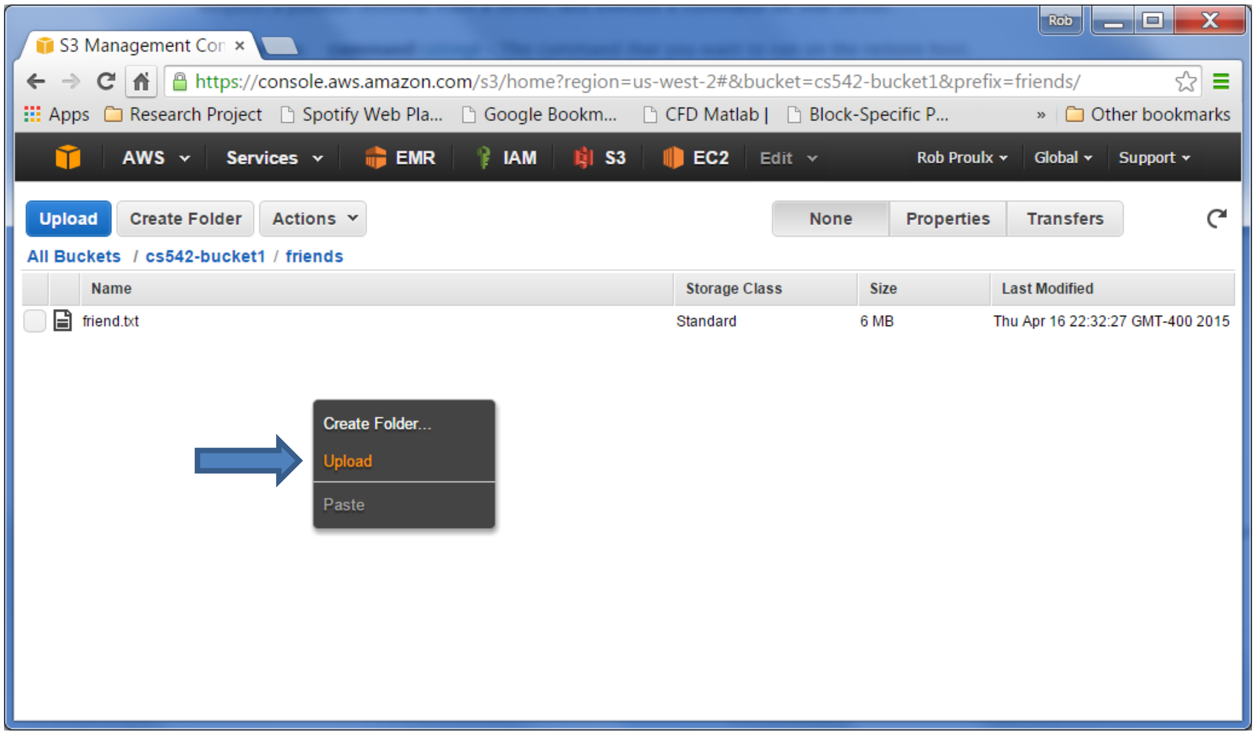
Several dataset were used to generate the country and city data, proper names, and words used in comments. Word.txt and propernames.txt come from /usr/share/dict in Macintosh System, OS X Yosemite, Version 10.10.2 and country_city.csv comes from worlddb - Open World Database - Google Project Hosting.

The end results of these functions are comma delimited text files.

3.2 Set Up AWS Backend

3.2.1 Upload Initial Data

With the test data generated, the next step in the process was to push this data up to the S3 datastore. This was done simply by using the S3 Web Interface. Each text file is placed into its own bucket, essentially a folder. Figure 6 below shows this interface. After selecting the correct bucket, a simple right click opens a pop-up menu with an 'Upload' option. In this case the bucket is called cs542-bucket1, sub-folder 'friends.'



Name	Storage Class	Size	Last Modified
friend.txt	Standard	6 MB	Thu Apr 16 22:32:27 GMT-400 2015

Figure 6—S3 Web Interface

Upload usertable.txt -> s3://cs542-bucket1/data/usertable.txt

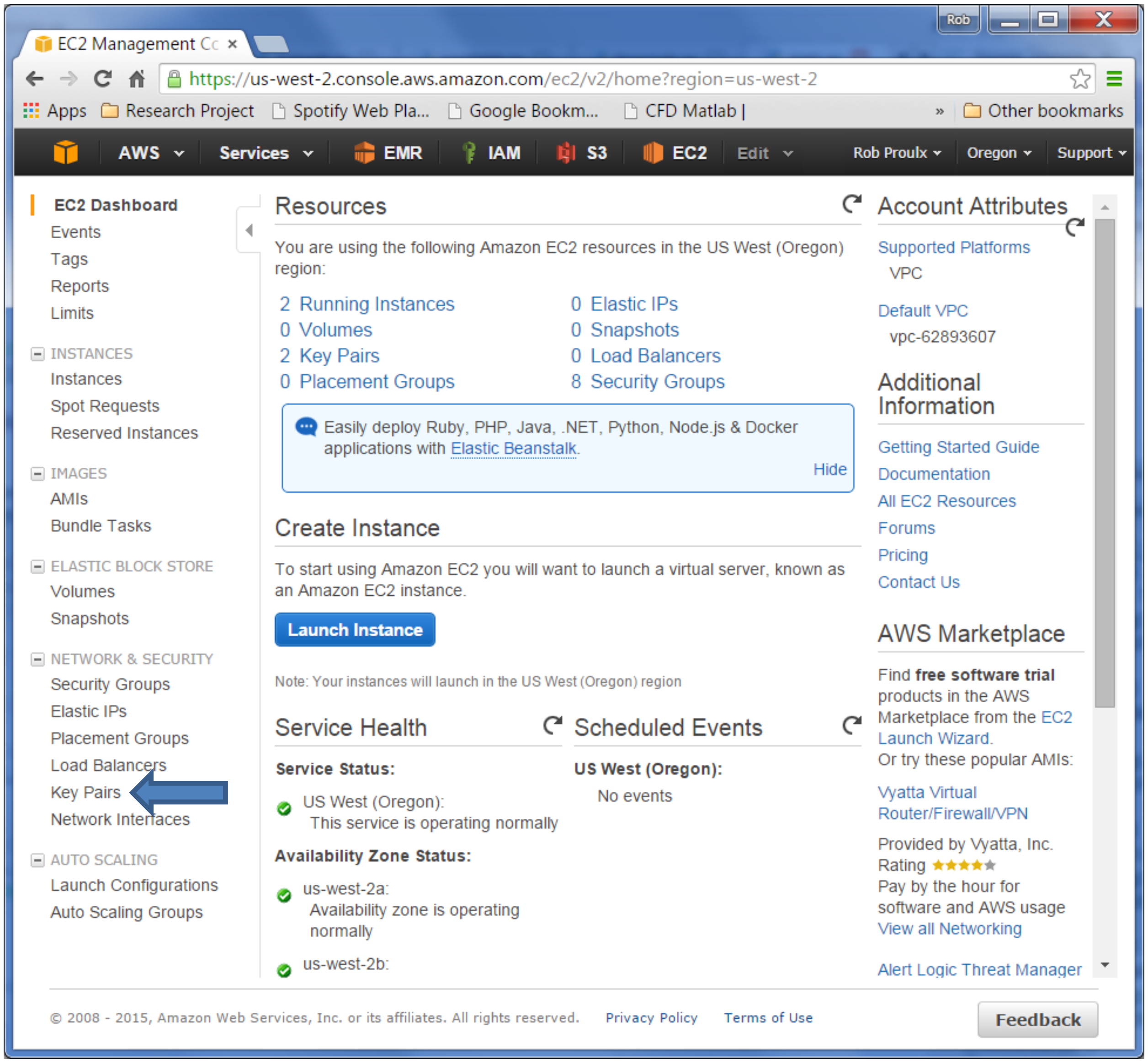
Upload friendtable.txt -> s3://cs542-bucket1/friends/friendtable.txt

Upload posts.txt -> s3://cs542-bucket1/posts/posts.txt

Upload posts.txt -> s3://cs542-bucket1/likes/likes.txt

3.2.2 Generate EC2 KeyPair

To access the main EMR through SSH connections on the client end, an RSA key file needs to be generated. Fortunately Amazon does this for us quite easily. Figure 7 shows the location of the Key Pairs Tool inside of the EC2 screen.



Resources

You are using the following Amazon EC2 resources in the US West (Oregon) region:

2 Running Instances	0 Elastic IPs
0 Volumes	0 Snapshots
2 Key Pairs	0 Load Balancers
0 Placement Groups	8 Security Groups

Easily deploy Ruby, PHP, Java, .NET, Python, Node.js & Docker applications with Elastic Beanstalk.

Hide

Create Instance

To start using Amazon EC2 you will want to launch a virtual server, known as an Amazon EC2 instance.

Launch Instance

Note: Your instances will launch in the US West (Oregon) region

Service Health

Service Status:

US West (Oregon):
This service is operating normally

Availability Zone Status:

us-west-2a:
Availability zone is operating normally

us-west-2b:

Scheduled Events

US West (Oregon):
No events

Account Attributes

Supported Platforms
VPC

Default VPC
vpc-62893607

Additional Information

Getting Started Guide
Documentation
All EC2 Resources
Forums
Pricing
Contact Us

AWS Marketplace

Find **free software trial** products in the AWS Marketplace from the EC2 Launch Wizard.
Or try these popular AMIs:

Vyatta Virtual Router/Firewall/VPN
Provided by Vyatta, Inc.
Rating ★★★★★
Pay by the hour for software and AWS usage
View all Networking

Alert Logic Threat Manager

© 2008 - 2015, Amazon Web Services, Inc. or its affiliates. All rights reserved. Privacy Policy Terms of Use Feedback

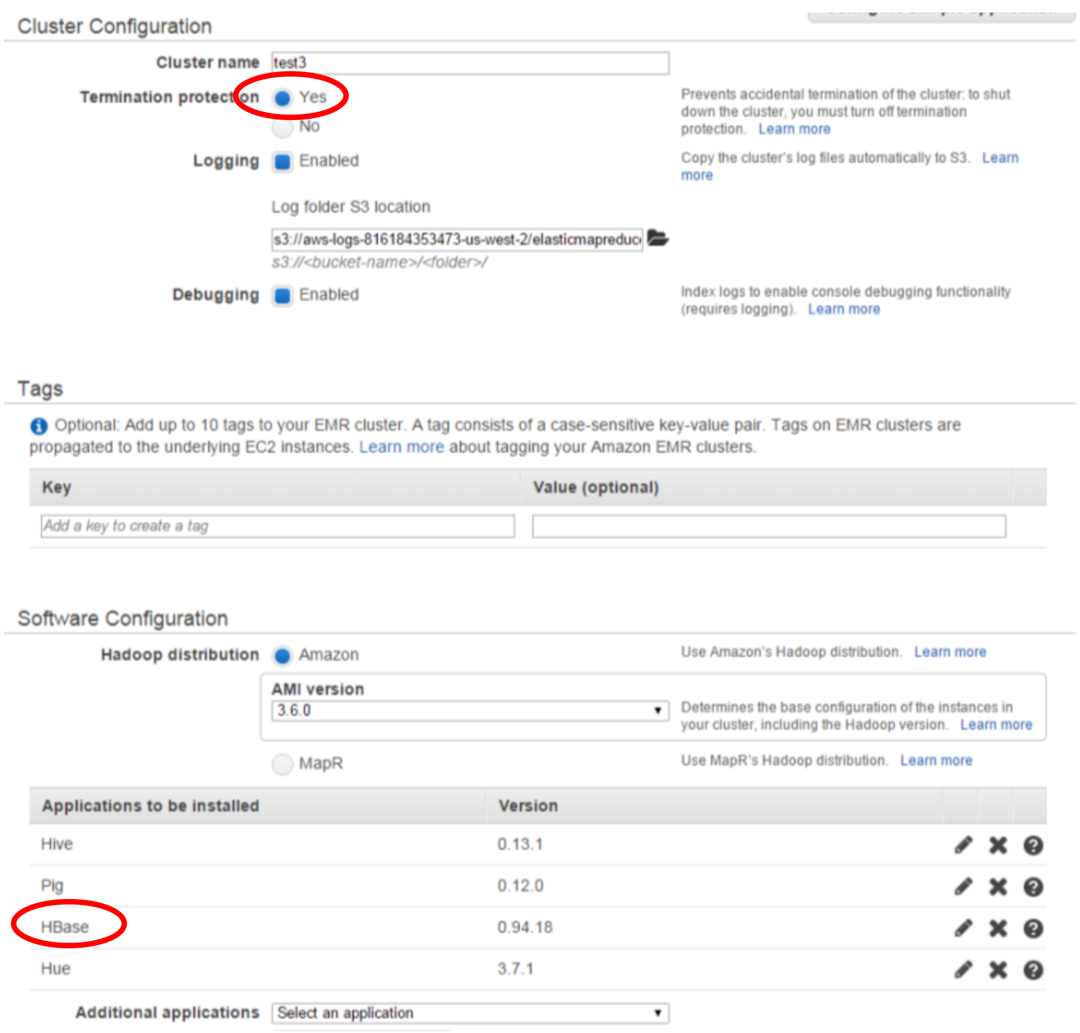
Figure 7—Key Pairs on the EC2 Menu

Invoking this command launches a simple wizard to create a key pair, and then to download a .pem file associated with this key. This file must be present on the client machine in order to connect to the EC2 instance.

3.2.3 Initiate EMR Instance

In the AWS console, open the EMR menu and select the Create Cluster button.

From there, termination protection must be on in order keep the EMR cluster running. HBase can be added to the applications to be installed if desired. Selecting the default AMI version for OS to run on the cluster should work well. Figure 8 shows these options.



Cluster Configuration

Cluster name	test3	
Termination protection	[✓ Yes] No	Prevents accidental termination of the cluster: to shut down the cluster, you must turn off termination protection. Learn more
Logging	[✓ Enabled]	Copy the cluster's log files automatically to S3. Learn more
	Log folder S3 location: s3://aws-logs-816184353473-us-west-2/elasticmapreduc s3://<bucket-name>/<folder>/	
Debugging	[✓ Enabled]	Index logs to enable console debugging functionality (requires logging). Learn more

Tags

Optional: Add up to 10 tags to your EMR cluster. A tag consists of a case-sensitive key-value pair. Tags on EMR clusters are propagated to the underlying EC2 instances. Learn more about tagging your Amazon EMR clusters.

Key	Value (optional)
Add a key to create a tag	

Software Configuration

Hadoop distribution	[✓ Amazon]	Use Amazon's Hadoop distribution. Learn more
	AMI version: 3.6.0	Determines the base configuration of the instances in your cluster, including the Hadoop version. Learn more
	MapR	Use MapR's Hadoop distribution. Learn more

Applications to be installed	Version
Hive	0.13.1
Pig	0.12.0
HBase	0.94.18
Hue	3.7.1

Additional applications: Select an application

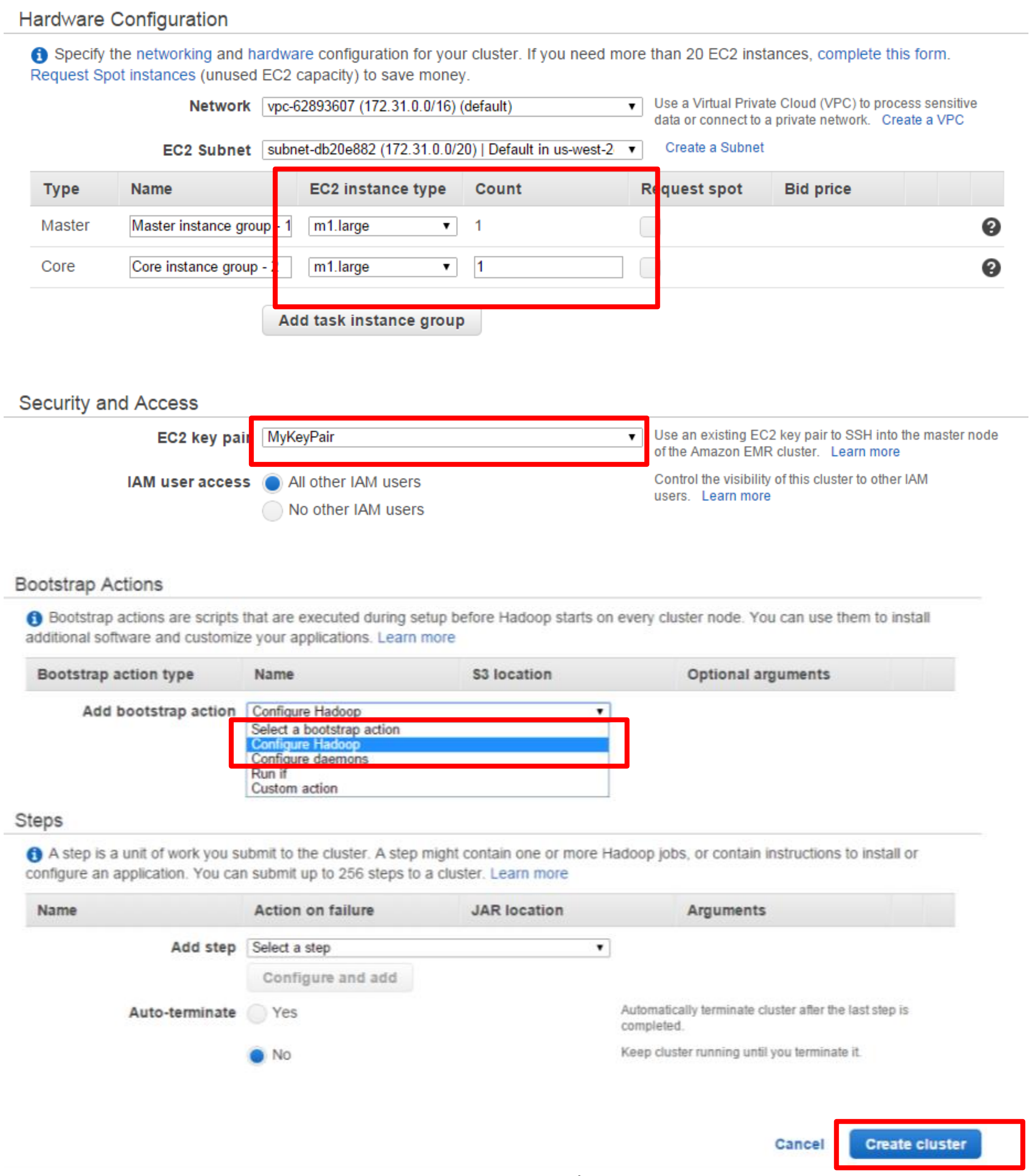
Figure 8—EMR Options 1

In this project development, only a single Master and Core node are spawned in the cluster. These nodes are set to be low performance m1.large instance types. These instance types are very economical, but low in performance. Selecting higher performing, but more expensive, instances as well as adding more core nodes can improve query speed.

Additionally, select the EC2 KeyPair previously (section 3.2.2) in order to connect to the nodes from the client.

Select 'Configure Hadoop' as a bootstrap action. Lastly click the create cluster button.

Figure 9 shows these EMR cluster options.



Hardware Configuration

Specify the networking and hardware configuration for your cluster. If you need more than 20 EC2 instances, complete this form. Request Spot instances (unused EC2 capacity) to save money.

Network vpc-62893607 (172.31.0.0/16) (default) — Use a Virtual Private Cloud (VPC) to process sensitive data or connect to a private network. Create a VPC

EC2 Subnet subnet-db20e882 (172.31.0.0/20) | Default in us-west-2 — Create a Subnet

Type	Name	EC2 instance type	Count	Request spot	Bid price
Master	Master instance group - 1	m1.large	1		
Core	Core instance group - 2	m1.large	1		

Security and Access

EC2 key pair MyKeyPair — Use an existing EC2 key pair to SSH into the master node of the Amazon EMR cluster. Learn more

IAM user access [✓ All other IAM users] No other IAM users — Control the visibility of this cluster to other IAM users. Learn more

Bootstrap Actions

Bootstrap actions are scripts that are executed during setup before Hadoop starts on every cluster node. You can use them to install additional software and customize your applications. Learn more

Bootstrap action type	Name	S3 location	Optional arguments

Add bootstrap action Configure Hadoop

Steps

A step is a unit of work you submit to the cluster. A step might contain one or more Hadoop jobs, or contain instructions to install or configure an application. You can submit up to 256 steps to a cluster. Learn more

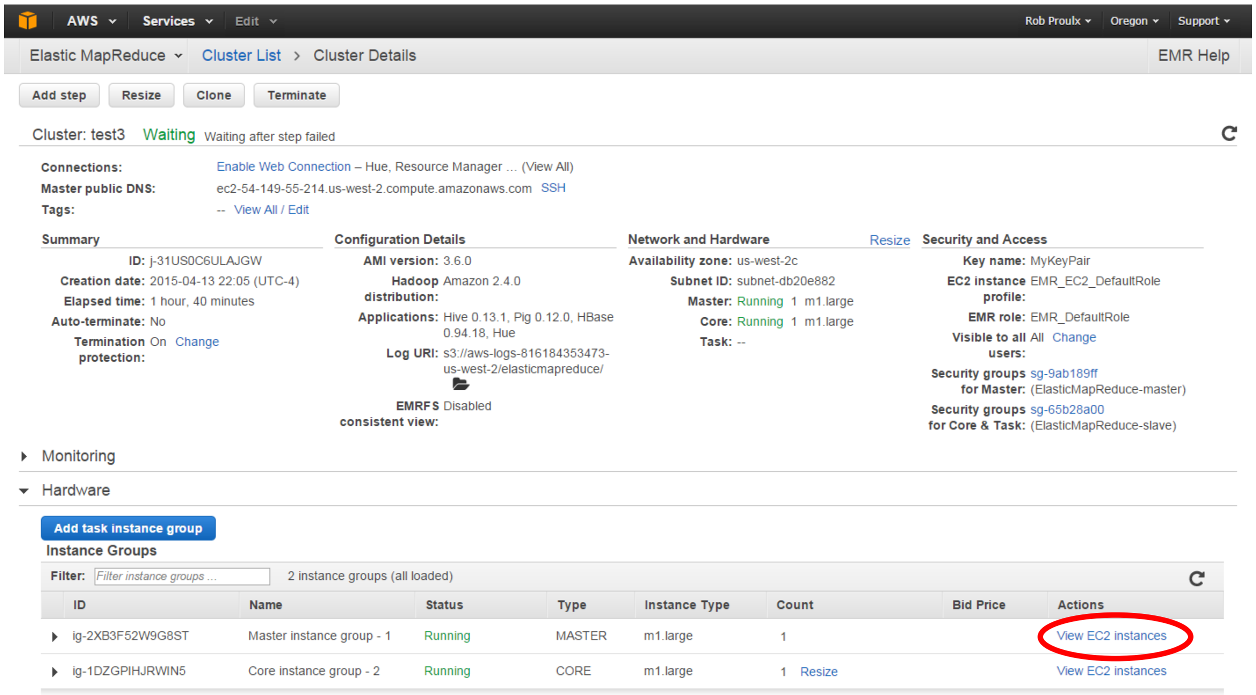
Name	Action on failure	JAR location	Arguments

Add step Select a step

Auto-terminate Yes — Automatically terminate cluster after the last step is completed. [✓ No] — Keep cluster running until you terminate it.

Figure 9—EMR Cluster Options 2

Now open the Cluster in the EMR->Cluster List Details, shown in Figure 10. Select View EC2 Instances on the Master Instance.



Cluster: test3 Waiting Waiting after step failed

Connections: Enable Web Connection – Hue, Resource Manager … (View All)
Master public DNS: ec2-54-149-55-214.us-west-2.compute.amazonaws.com SSH
Tags: -- View All / Edit

Summary
ID: j-31US0C6ULAJGW
Creation date: 2015-04-13 22:05 (UTC-4)
Elapsed time: 1 hour, 40 minutes
Auto-terminate: No
Termination protection: On Change

Configuration Details
AMI version: 3.6.0
Hadoop distribution: Amazon 2.4.0
Applications: Hive 0.13.1, Pig 0.12.0, HBase 0.94.18, Hue
Log URI: s3://aws-logs-816184353473-us-west-2/elasticmapreduce/
EMRFS consistent view: Disabled

Network and Hardware Resize
Availability zone: us-west-2c
Subnet ID: subnet-db20e882
Master: Running 1 m1.large
Core: Running 1 m1.large
Task: --

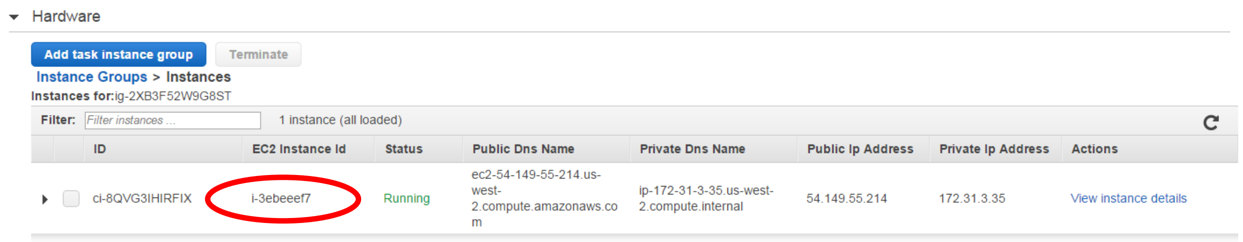
Security and Access
Key name: MyKeyPair
EC2 instance profile: EMR_EC2_DefaultRole
EMR role: EMR_DefaultRole
Visible to all users: All Change
Security groups for Master: sg-9ab189ff (ElasticMapReduce-master)
Security groups for Core & Task: sg-65b28a00 (ElasticMapReduce-slave)

Monitoring
Hardware
Add task instance group
Instance Groups
Filter: Filter instance groups … 2 instance groups (all loaded)

ID	Name	Status	Type	Instance Type	Count	Bid Price	Actions
ig-2XB3F52W9G8ST	Master instance group - 1	Running	MASTER	m1.large	1		View EC2 instances
ig-1DZGPIHJRWIN5	Core instance group - 2	Running	CORE	m1.large	1 Resize		View EC2 instances

Figure 10—EMR Cluster Details.

Locate the EC2 instance ID of the Master Instance, as shown in Figure 11.



Instance Groups > Instances
Instances for:ig-2XB3F52W9G8ST
Filter: Filter instances … 1 instance (all loaded)

ID	EC2 Instance Id	Status	Public Dns Name	Private Dns Name	Public Ip Address	Private Ip Address	Actions
ci-8QVG3IHIRFIX	i-3ebeeef7	Running	ec2-54-149-55-214.us-west-2.compute.amazonaws.com	ip-172-31-3-35.us-west-2.compute.internal	54.149.55.214	172.31.3.35	View instance details

Figure 11—EC2 Master Instance Details

With this instance ID, we can use the client-side python functions to do our work.

3.3 Client-Side Functions

3.3.1 Function list

Several Python functions comprise the Client-Side part of the project. They are located on github at: https://github.com/robproulx/CS542

The Python files and their key functions are:

- botossh.py: S3 and EC2 Connection and Instance management through the boto library ([https://pypi.python.org/pypi/boto)](https://pypi.python.org/pypi/boto)
    - `createSSHClient(instanceID (String) ,user_name='hadoop' (String)– Returns ssh_client connection object to EC2 instance`
    - `createS3Connection()–Returns object to S3`
- constants.py: storage of several constants used throughout for convenience
- hivecmds.py: wrapper for the various Hive queries.
    - `reattachDatabases(ssh_client object)`
        - no Returns
        - Used to reattach the database tables to a newly spawned Hive session
    - Various other wrappered function detailed in section 3.3.3
    - All other hivecmds.py functions return:
        - `r (list of dict objects for parsing),`
        - `pretty (list of strings for pretty console print)`
- main.py: Python script to test functionality. Calls hivecmds.reattachDatabases and then the various functions as shown in Section 3.3.3. Output is printed to Python console

The hive commands are performed by using the following syntax: `ssh_client.run('hive –e "hive_command_string;"')`. Invoking the `.run()` method of the ssh_client object injects a linux bash command to the command line. The 'hive –e "hive_command_string"' syntax initiates hive, runs the "hive_command_string" within hive, and then exits Hive. The `.run()` method then returns the linux console result of the command to the client. Loading and unloading Hive is not a realistic production solution, since a time penalty is paid for loading and unloading.

3.3.2 CREATE TABLES

One limitation to using Amazon's EMR clusters is that of trying to keep cost low. The cost of maintaining a persistent EBS datastore, or keeping the EMR cluster running over weeks of testing and development would be prohibitive for this project. Therefore, when the EMR cluster loads anew for a development session, it must "relearn" the database tables present in the previous sessions.

The basic syntax used to reconnect the EMR cluster Hive instance to the S3 data after restart is CREATE TABLE. Note: these CREATE TABLE Hive commands are all wrappered in the `hivecmds.reattachDatabases` function

Create users table from the s3://cs542-bucket1/data/ location.

```
CREATE TABLE user(
    > id INT,
    > name STRING,
    > gender STRING,
    > birthday DATE,
    > country STRING,
    > city STRING)
    > ROW FORMAT DELIMITED FIELDS TERMINATED BY ','
    > LOCATION 's3://cs542-bucket1/data/';
```

Create friends table from the s3://cs542-bucket1/friends/ location.

```
CREATE TABLE friend(
    > id1 INT,
    > id2 INT)
    > ROW FORMAT DELIMITED FIELDS TERMINATED BY ','
    > LOCATION 's3://cs542-bucket1/friends/';
```

Create posts table from the s3://cs542-bucket1/posts/ location.

```
CREATE TABLE post(
    > id INT,
    > uid INT,
    > content STRING,
    > postdate TIMESTAMP)
    > ROW FORMAT DELIMITED FIELDS TERMINATED BY ','
    > LOCATION 's3://cs542-bucket1/posts/';
```

Create posts table from the s3://cs542-bucket1/likes/ location.

```
CREATE TABLE likes(
    > messageid INT,
    > userid INT,
    > likedate TIMESTAMP)
    > ROW FORMAT DELIMITED FIELDS TERMINATED BY ','
    > LOCATION 's3://cs542-bucket1/likes/';
```

Note: the EXTERNAL keyword could be added after TABLE to force Hive to treat the S3 data as a foreign datastore. Additionally PARTITION BY keywords could be added to alert Hive to the fact that data will be partitioned across several subfolders in the S3 buckets. This was not necessary for the test cases examined.

3.3.3 Run Queries

Below are the example functions, the syntax within the main.py script and the Python console results. In these examples the variable 'cl' is the ssh_client object connected to the EC2 master node instance.

3.3.3.1 Find Users With Similar Names

Function: Find all users with names containing 'Manny'

Syntax: `hivecmds.searchUserByName(cl,"Manny")`

Result:

```
Users Matching 'Manny' in their name
id. Name
2. Manny Joyce
```

1959. Manny Brian
2143. Manny Miriamne
2228. Alain Manny
2614. Manny Jean-Pierre
2630. Manny Bud
2677. Michelle Manny
2983. Manny Sandeep
3826. Manny Larry
4184. Manny Earl
4845. Alastair Manny
4978. Manny Shakil
5609. Pierette Manny
6145. Manny Allan
6860. Max Manny
6893. Will Manny
7092. Manny Del
7279. Manny Ahmet
7490. Alain Manny
8828. Manny Leon
9110. Trent Manny
9208. Manny Carolyn
9386. Manny Sassan
10871. Barrio Manny
12274. Manny Heather
12432. Manny Caroline
13170. Manny Brad
13532. Saad Manny
14264. Manny Hein
14825. Teri Manny
16852. Manny Owen
16892. Manny Serdar
17105. Manny Pim
17816. Manny Milner
19225. Manny Guillermo
20339. Willie Manny
20503. Manny Juergen
23007. Manny Clyde
23093. Manny Linley
23557. Dawn Manny
23791. Tuna Manny
25082. Elliott Manny
26323. Manny Sundar
26561. Manny Conrad
26907. Dory Manny
27020. Manny Julie
27105. Manny Giovanni
27433. Manny Dewey
27501. Manny Oliver
27567. Manny Syun
29913. Manny Sassan
31591. Linder Manny
31677. Brodie Manny
31958. Nicolas Manny
32276. Steen Manny
32479. Manny Orville
33231. Kurt Manny
34195. Manny Ragnar

34892. Rex Manny
35508. Manny Ramadoss
35998. Ernst Manny
36096. Manny Betty
37548. Ravi Manny
38256. Manny Griff
38905. Andrew Manny
40009. Chuck Manny
40486. Guido Manny
40657. Ritalynne Manny
41394. Luis Manny
41912. Manny Naomi
43031. Mahesh Manny
43495. Jiri Manny
45177. Manny Leonard
45628. Irfan Manny
46064. Manny Betsy
46539. Seth Manny
46738. Manny Giles
47005. Jim Manny
47074. Manny Kurt
47122. Manny Naim
49319. Antonella Manny
49679. Sumitro Manny
50223. Anthony Manny
51468. Tracy Manny
51533. Manny Duane
53351. Julie Manny
54038. Jelske Manny
54748. Manny Jess
55043. Manny Lex
56326. Manny Carole
56338. Marshall Manny
56490. Konrad Manny
56594. Manny Cindie
57681. Manny Anton
57709. Hartmann Manny
57969. Manny Hunter
58005. Pablo Manny
58942. Lea Manny
59062. Susanne Manny
61133. Manny Hal
61791. Manny Emmett
62646. Alf Manny
62767. Manny Mary
63112. Manny Tracy
63845. Ti Manny
64359. Po Manny
64408. Manny Cathy
65476. Terry Manny
65589. Marcos Manny
65959. Leads Manny
66297. Manny Stephe
67003. Manny Wolf
67463. Heather Manny
67817. Manny Stevan
67842. Robin Manny

```
68333. Stefan Manny
69115. Manny Hon
69152. Manny Teruyuki
69161. Shatter Manny
69455. Jacob Manny
69660. Samuel Manny
69833. Manny Hwa
70417. Manny Ted
70971. Roy Manny
70981. Manny Swamy
71358. Manny Shirley
71404. Manny Lin
72551. Jwahar Manny
73949. Rainer Manny
74326. Hazel Manny
76002. Manny Elric
76160. Loukas Manny
79162. Ramesh Manny
79607. Manny Rolf
79957. Manny Rudolph
80573. Brender Manny
80901. Alex Manny
81503. Manny Matthieu
81875. Manny Briggs
83128. Manny Mosur
83321. Manny Panos
83344. Manny Timo
83494. Gary Manny
84448. Leigh Manny
85618. Manny Carole
86650. Dalton Manny
87247. Manny Brad
89206. Vincent Manny
89496. Manny Meehan
90561. Ginny Manny
90638. Manny Mary
90772. Eli Manny
90953. Manny Roxane
92669. Tad Manny
92882. Manny Gail
93208. Manny Nancy
93379. Manny Panacea
93568. Joel Manny
93926. Jaime Manny
94511. Teriann Manny
95800. Hirofumi Manny
96418. Manny Tolerant
98801. Judith Manny
99409. Manny Raman
99744. Manny Annie
```

3.3.3.2 Find Friends of User by UserID

Function: Find all friends of user.uid=2

Syntax: `hivecmds.listFriends(cl,uid=2)`

Result:

```
Friends of Manny Joyce:
Tai Saqib
Pradeep Trent
Ram Rahul
Johnnie Liz
Nils Shahid
Dale Pedro
Valeria Judith
```

3.3.3.3 Find All Posts By UserID

Function: Find all of the posts and timestamps by UserID ordered newest to oldest.

Syntax: `hivecmds.getUserPostsByUserID(cl,uid=2)`

Result:

```
All Messages by Manny Joyce
besqueeze irregularize unsutured posted on 2011-02-10 15:48:44
tearful bezzi scuppet unrosined posted on 2006-10-05 06:49:48
archontic posted on 2005-04-07 21:31:15
Nipponism anadenia posted on 2004-08-27 22:10:29
```

Note: the comments are just a random group of words, not expected to make sense.

3.3.3.4 Find All Likes of Message by ID

Function: Find all of the users who liked a post and the timestamp of the like, ordered by newest to oldest.

Syntax: `hivecmds.getLikesOfPostsByMsgID(cl,msg_id)`

Result:

```
Likes of Post id.5 Nipponism anadenia posted on 2004-08-27 22:10:29
by Ram Rahul on 2008-09-30 00:22:19
by Dale Pedro on 2008-08-17 06:27:24
by Johnnie Liz on 2007-09-10 06:37:02
by Nils Shahid on 2005-09-27 23:01:27
by Tai Saqib on 2002-11-05 03:34:03
```

3.3.3.5 Find All Users from a City

Function: Find all of the users in a given city by city name

Syntax: `hivecmds. searchUserByCity(cl,city="Burlington")`

Result:

```
Users from Burlington:
107. Jacob Jayant
517. Mechael Hector
902. Jagath Syun
1922. Cristina Rudolf
```

4326. Ian Rudy
5061. Agatha Rand
5554. Gil Frederic
6025. Stefan Subra
6047. Naoto Coleen
6092. Milner Toft
6694. Jesse Jerald
12483. Cathryn Ramanan
12678. Alf Spock
13320. Cindy Saul
13395. Pierce Edmund
14914. Roxane Skef
17402. Sriram Thuan
18480. Ahmet Brandi
18910. Art Kyu
19995. Noam Lanny
20635. Shakil Page
20849. Marla Murthy
22010. Elias Izchak
22628. Alain Sanity
22812. Edwin Murray
23180. Woody Marcia
23376. Clifford Knapper
26329. Matti Amir
26710. Jane Barton
28184. Liyuan Debi
30180. Gregg Leif
30767. Bruce Knute
31877. Huashi Darrell
33329. Jayesh Ted
34171. Cris Seenu
34534. Olson Ted
34756. Vic Lance
35173. Amanda Perry
35675. Gail Geoffrey
36154. Russ Benson
38909. Liber Lila
39800. Elsa Harris
40836. Andrew Herman
41598. Ritchey Rajesh
42034. Terrence Tareq
43038. Rathnakumar Ram
43385. Wilmer Tovah
44457. Rod Randall
45035. Patrick Milner
46135. Piotr Dennis
48175. Shutoku Gideon
48508. Ricardo Gretchen
48578. Ross Liza
49362. Audrey Sriram
50746. Heidi Ragnar
52224. Carter Roman
53903. Clark Devon
54336. Mikey Dan
58287. Mohammad Leif
59236. Clark Andrew
61174. Rodent Frances

```
61176. Jarl Vance
61830. Graeme Knudsen
62201. Norm Mick
63682. Alan Grant
63733. Jesse Santa
64527. Kenneth Shai
64832. Lloyd Jingbai
65424. Turkeer Francisco
65590. Israel Marcel
65929. Cristopher Theo
66223. Sundar Benjamin
66479. Rolf Tovah
67660. Linley Ned
71646. Tobias Shari
71739. Fletcher Jeannie
71872. Hector Jerrie
73796. Laurianne Sandy
76890. Rodney Olson
77225. Real Roberta
77251. Granville Jenine
78994. Kristi Sergio
80075. Sally Jakob
82262. Tad Danielle
82671. Stewart Vishal
83445. Vincenzo Kylo
83781. Kevin Gabriel
83991. Lord Srinivas
85031. Sundar Soohong
85884. Marcia Tammy
87089. Florian Irving
87938. Victoria Blaine
89014. Chet Kuldip
91897. Ernest Edmund
92615. Bruce Irwin
93617. Roxanne Hitoshi
93646. Marek Vadim
93708. Rusty Clayton
96075. Lorraine Drew
97041. Po Sandip
98070. Jill Ralf
```

3.3.3.6 Find Posts with a Keyword

Function: Search all posts for content matching a keyword.

Syntax: `hivecmds.getPostsByKeyword(cl,keyword='save')`

Result:

```
Posts Containing Keyword 'save'
id. 15 'save' by Heinz Gregor on 2001-09-30 05:10:00
id. 2060 'opianic valeryl unexplained lifesaver' by Philip Archie on 2011-04-
29 21:32:30
id. 6065 'casave spicing slubberdegullion unbendableness squirtiness
centauress' by Pratap Ramanan on 2002-08-24 22:43:07
id. 16232 'lifesaver thoughted preinterfere purposefully' by Sid Stewart on
2001-04-17 17:29:12
```

```
id. 21645 'save' by Rob Johann on 2004-06-05 15:22:58
id. 46772 'amyotrophy flatteringness craft supracaudal saveloy vasquine
picketer' by Lila Cole on 2000-07-09 02:59:45
id. 48612 'aleuronat proverbiology unsaveable doodle antistrophon
velellidous' by Lin Valeria on 2004-01-03 04:44:24
id. 49618 'undulated unsaved' by Dick Martha on 2008-04-17 21:08:09
id. 60916 'roritorious naphthalidine concubine casave quadrisulcated derat'
by Sofia Debi on 2000-08-01 07:31:08
id. 63902 'placate casave salmis' by Vicky Christofer on 2011-07-23 21:54:59
id. 72927 'unsaveable unauspiciousness' by Dale Ralf on 2014-03-17 02:06:59
id. 81409 'saver colonizer plumasite brushite divinable' by Alvin Barney on
2014-05-01 07:31:58
id. 95273 'conker saver circummeridian' by Pieter Elvis on 2013-09-16
15:08:57
id. 97523 'notable tympanosis Monozoa saved' by Cathy Lori on 2011-04-21
07:16:09
id. 98111 'mossed exfoliative prolongate Tad psychrophile spangler timesaver'
by Sergei Tran on 2011-03-09 16:33:46
id. 98696 'unsaveable pyritohedral sable hamulus mimmocking unsatirize' by
Reinhard Ethan on 2001-01-08 08:00:53
id. 102858 'forcefulness disfigurement unsaved autogenetically' by Knut
Rebecca on 2013-11-14 18:54:50
id. 105524 'timesaver potted paleoglyph seismograph mushaa' by Hein Daren on
2012-06-16 19:32:07
id. 112416 'phygogalactic withsave sculler Colosseum' by Patrick Milner on
2014-08-03 21:36:38
id. 120553 'idioelectric alderwoman unsepulchered oversave quizzicalness
fustian' by Case Boyce on 2010-01-09 11:41:40
id. 122581 'amphichroic saver nonsectarian extern overinsist' by Jem Bonnie
on 2012-01-05 19:43:01
id. 124118 'Osage timesaver highliving entreat loverhood overseam resection'
by Albert Linder on 2008-06-06 18:07:00
id. 131456 'twaddleize columniferous unsaved today' by Rabin Suwandi on 2003-
03-02 06:17:28
id. 140275 'microgranular dorsolumbar lateral break overuse hopoff unsaved'
by Billy Vivek on 2007-11-05 16:28:13
id. 141017 'equatorwards Oxytropis save' by Saiid Kelvin on 2004-02-17
11:56:15
id. 154036 'sledgemeter uniflowered lifesaver laccol' by Murat Simon on 2011-
11-08 18:15:55
id. 155713 'meddlecome interparietal overthrower casave ephemeron plainish
aortarctia' by Peggy Pilar on 2012-04-23 19:41:34
id. 156609 'shoddywards terraqueous unvaluableness lifesaver recentre
drilling homology' by Sorrel Ramon on 2011-08-29 20:08:01
id. 159753 'saved sassafras procritic piperylene merribauks' by Gunter
Gabriel on 2002-03-02 18:35:28
id. 167226 'environmentally sharkship casave overrunningly inoma' by Melinda
Soohong on 2002-01-26 23:21:05
id. 175159 'timesaver eupatoriaceous Lacerta' by Lenora Nou on 2014-10-06
01:42:23
id. 179228 'gynocracy timesaver exemplificational dysphagia panning
diversiflorous dookit' by Sehyo Beverly on 2009-03-13 12:23:04
id. 185366 'Vespertilio casave' by Lenora George on 2005-05-01 00:18:19
id. 190688 'inteind Temne god joky withsave phyllocyanin' by Ethan Raja on
2012-08-14 15:51:48
id. 200923 'saveloy metapsychism' by Lila Ramon on 2005-06-29 20:24:21
id. 202687 'isatate gregarious nonunionism save caries sunburntness
uncommunicating' by Jayesh Barbara on 2014-02-22 22:32:52
id. 206641 'saved whist Sphaeriales' by Jon Hiroyuki on 2014-09-06 18:46:35
id. 207022 'yellowammer groundplot arrayment lifesaver indirected' by Ragnar
Jorge on 2002-03-25 16:28:58
id. 214616 'nullipennate relocator casave crackerjack' by Raja Todd on 2011-
09-20 04:44:39
id. 215626 'subdeaconess cellulicidal confluently kickless fishbed casave' by
Jayesh Arnold on 2011-05-08 05:11:27
id. 224021 'foundationless metropolitanate Indophile bonewort withsave
homogamy thermoscopic' by Lea Jakob on 2005-01-22 07:16:18
id. 226689 'prolongably saver organizable blunderingly pearten bakeoven
inextinguishable' by Naresh Pierce on 2010-10-15 03:00:22
id. 242878 'saved broidery trichopathic quintillionth ego' by Pantelis Karen
on 2005-07-23 21:40:47
```

3.4 Comments on Adding

Naturally, a real system must not only query data, but add new data. Unfortunately, the Hive instances supported by the AWS EMR clusters do not support a direct SQL-like INSERT Method.

A proposed workaround would be:

1. Determine new data, for example a user would be [USER_ID, USER_NAME, GENDER, BIRTHDAY, COUNTRY, CITY]
2. Write this comma separate string to a file in a temp bucket in S3
3. Create a table in the EMR connected to this bucket:

   ```
   CREATE TABLE temp(
       > id INT,
       > name STRING,
       > gender STRING,
       > birthday DATE,
       > country STRING,
       > city STRING)
       > ROW FORMAT DELIMITED FIELDS TERMINATED BY ','
       > LOCATION 's3://cs542-bucket1/temp/';
   ```
4. User the INSERT INTO table SELECT methodology: `INSERT INTO TABLE user SELECT * FROM temp;`

This method should allow of insertion of new elements into the existing tables. In this implementation, various S3 bucket and key management commands would need to be negotiated to place the temp data into the store.

3.5 Comments on Implementation

The above workflow presented has many boot-strap type processes, somewhat jumping from platform to platform. In the ideal implementation, deemed well beyond the scope for this project, a dedicated Hadoop hardware cluster would be procured, and the data would remain persistent upon it directly in HDFS, rather than relying on the EMR to S3 connection. Machines could be added to the cluster, and data could be partitioned in the HDFS framework to manage size. Though speed was not studied in this project, it would be greatly increased than seen in these examples.

4 Conclusions

A simplified Hadoop based framework to perform the basic Facebook functionality is shown. Simulated user, post, like, share and friend data was created. This data was stored on Amazon's S3 cloud. A local set of client functions used an Amazon EMR cluster to connect to, query and manipulate this data.

This functionality does work, and suggests another viable pathway that Facebook could have chosen, instead of MySQL, for their core datastore. Of course, Facebook's decision to use MySQL predates the development of Hive and HBase as Hadoop data storage options, so it's difficult to be critical of their decision. However, now in 2015, a different solution could have been taken.

5 Acknowledgements

For the Python front end, the Boto function libraries are used to connect to and manage the Amazon S3 and EMR. This software is available under the MIT License. More information about the package can be found at: https://pypi.python.org/pypi/boto