VERSION 1

FEBRUARY 10, 2018

PRESENTED BY: SPIWAK, LARRY

Problem Statement

Attempt faster data loads of millions of records into Mongo DB utilizing the Hadoop MapReduce infrastructure built into the HD Insight cluster. The RXBC MRDD and PBI extract at RelayHealth produces daily files ranging from 7 million to 23 million or more data documents per file. A single-threaded load would take too long, but Mongo can accommodate multiple threads connected to it, loading data in parallel to speed up loads by orders of magnitude. Compare this performance to RelayHealth Data Service's grid-computing environment's nominal performance.

Technology Overview

Microsoft Azure HD Insight / Hadoop MapReduce

Cluster Type, HDI Version

Hadoop on Linux (HDI 3.6)

Head Nodes (D12 v2 x2) Worker Nodes (D4 v2 x4) – 40 cores

Cosmos DB / Mongo DB API

prod_mrdd / MRDD collection @ 1000 RU/second

Overview of steps

1. Configured environment (installed HD Insight, Cosmos DB)
2. Configured running environment (Java code base, Java libraries, Java compile script, Hadoop run script) by cloning from git repository https://github.com/lumkichi/DeepAzure_Project
3. Obtained data from RelayHealth DataServices (available in GitHub)
4. Copy out connection string from Azure Portal and ReCompile code
5. Load the data into HDFS
6. Run the MongoLoader.jar
7. View data load success in RoboMongo 3T

Data Set

The Data set originates daily and internally at RelayHealth DataServices' RXBC daily batch process. Known as the "Most Recently Dispensed Drug for the Most Frequently Dispensed Quantity" or "MRDD" for short, it is a collection of some 240 million records of every drug (by therapeutic class) dispensed at every pharmacy that has come through RelayHealth's switching network. It is a flat-file of 15 fields containing the most recent pricing information for nearly all drugs across all pharmacies. Data has been uploaded to GitHub

Software & Libraries

- Java JDK 1.8 (installed), Hadoop HDI 3.6 (installed)
- MongoDB Java Driver https://oss.sonatype.org/content/repositories/releases/org/mongodb/mongodb-driver/3.6.1
- SLF4J No Op library http://central.maven.org/maven2/org/slf4j/slf4j-nop/1.7.25/
- BitVise SSH Client https://www.bitvise.com/ssh-client-download
- RoboMongo 3T https://robomongo.org/download

Lessons Learned, Pros / Cons

- RelayHealth Data Services currently has a grid infrastructure with 2 control nodes and over 40 remote servers with a total of 1250 cores.
- While a parallel computing infrastructure has been developed in-house and put to good use through, it was well-worth investigating if an off-the-shelf technology such as Hadoop in the Cloud could be leveraged to handle the work we currently do in-house.
- Surprisingly easy (for a seasoned Java developer like me) to utilize.
- However because of the way MapReduce breaks up the file into chunks determined by size rather than row count, the data fed to the MongoMap was sub-optimal:
    - Could not take advantage of more parallel threads
    - Could not take advantage of bulk insert operations which are much faster than single inserts
- MongoDB was also restricted at 1000 RU/s, which also led to suboptimal operation
- However programs must be written in Java with the Hadoop libraries compiled in, requires dedicated rewrite instead of simply "porting over" Data Services code.
- Demonstrated that we can successfully load Mongo Data in Parallel.

YouTube URL's & Git Repository

- Two minute (short): https://www.youtube.com/watch?v=4XZVUQicuIM
- 15 minutes (long): https://www.youtube.com/watch?v=YHRTkNVww3k&t=617s
- GitHub Repository with all artifacts: https://github.com/lumkichi/DeepAzure_Project

Prepare the HD Insight and MongoDB Template Files

```
{
    "$schema": "http://schema.management.azure.com/schemas/2014-04-01-preview/deploymentTemplate.json#",
    "contentVersion": "0.9.0.0",
    "parameters": {
        "clusterName": {
            "type": "string",
            "defaultValue": "lummyhd",
            "metadata": {
                "description": "The name of the HDInsight cluster to create."
            }
        },
        "clusterLoginUserName": {
            "type": "string",
            "defaultValue": "admin",
            "metadata": {
                "description": "These credentials can be used to submit jobs to the cluster and to log into cluster
dashboards."
            }
        },
        "clusterLoginPassword": {
            "type": "securestring",
            "defaultValue": "somePassword#424",
            "metadata": {
                "description": "The password must be at least 10 characters in length and must contain at least one
digit, one non-alphanumeric character, and one upper or lower case letter."
            }
        },
        "location": {
            "type": "string",
            "defaultValue": "eastus",
            "metadata": {
                "description": "The location where all azure resources will be deployed."
            }
        },
        "clusterVersion": {
            "type": "string",
            "defaultValue": "3.6",
            "metadata": {
                "description": "HDInsight cluster version."
            }
        },
        "clusterWorkerNodeCount": {
            "type": "int",
            "defaultValue": 4,
            "metadata": {
                "description": "The number of nodes in the HDInsight cluster."
            }
        },
        "clusterKind": {
            "type": "string",
            "defaultValue": "HADOOP",
            "metadata": {
                "description": "The type of the HDInsight cluster to create."
            }
        },
        "sshUserName": {
            "type": "string",
            "defaultValue": "sshuser",
            "metadata": {
                "description": "These credentials can be used to remotely access the cluster."
            }
        },
        "sshPassword": {
            "type": "securestring",
```

```
            "defaultValue": "somePassword#424",
            "metadata": {
                "description": "The password must be at least 10 characters in length and must contain at least one
digit, one non-alphanumeric character, and one upper or lower case letter."
            }
        }
    },
    "resources": [
        {
            "apiVersion": "2015-03-01-preview",
            "name": "[parameters('clusterName')]",
            "type": "Microsoft.HDInsight/clusters",
            "location": "[parameters('location')]",
            "dependsOn": [],
            "properties": {
                "clusterVersion": "[parameters('clusterVersion')]",
                "osType": "Linux",
                "tier": "standard",
                "clusterDefinition": {
                    "kind": "[parameters('clusterKind')]",
                    "configurations": {
                        "gateway": {
                            "restAuthCredential.isEnabled": true,
                            "restAuthCredential.username": "[parameters('clusterLoginUserName')]",
                            "restAuthCredential.password": "[parameters('clusterLoginPassword')]"
                        }
                    }
                },
                "storageProfile": {
                    "storageaccounts": [
                        {
                            "name": "lummystorage.blob.core.windows.net",
                            "isDefault": true,
                            "container": "lummyhd-2018-02-07t02-25-48-422z",
                            "key": "[listKeys('/subscriptions/19ce215b-e250-49ce-a40f-
e3efd1217efc/resourceGroups/lummyrg/providers/Microsoft.Storage/storageAccounts/lummystorage', '2015-05-01-
preview').key1]"
                        }
                    ]
                },
                "computeProfile": {
                    "roles": [
                        {
                            "name": "headnode",
                            "minInstanceCount": 1,
                            "targetInstanceCount": 2,
                            "hardwareProfile": {
                                "vmSize": "Standard_D12_V2"
                            },
                            "osProfile": {
                                "linuxOperatingSystemProfile": {
                                    "username": "[parameters('sshUserName')]",
                                    "password": "[parameters('sshPassword')]"
                                }
                            },
                            "virtualNetworkProfile": null,
                            "scriptActions": []
                        },
                        {
                            "name": "workernode",
                            "minInstanceCount": 1,
                            "targetInstanceCount": 4,
                            "hardwareProfile": {
                                "vmSize": "Standard_D4_V2"
                            },
                            "osProfile": {
                                "linuxOperatingSystemProfile": {
                                    "username": "[parameters('sshUserName')]",
                                    "password": "[parameters('sshPassword')]"
                                }
                            },
                            "virtualNetworkProfile": null,
                            "scriptActions": []
                        },
                        {
                            "name": "zookeepernode",
                            "minInstanceCount": 1,
                            "targetInstanceCount": 3,
```

```
                            "hardwareProfile": {
                                "vmSize": "Small"
                            },
                            "osProfile": {
                                "linuxOperatingSystemProfile": {
                                    "username": "[parameters('sshUserName')]",
                                    "password": "[parameters('sshPassword')]"
                                }
                            },
                            "virtualNetworkProfile": null,
                            "scriptActions": []
                        }
                    ]
                }
            }
        }
    ]
}
```

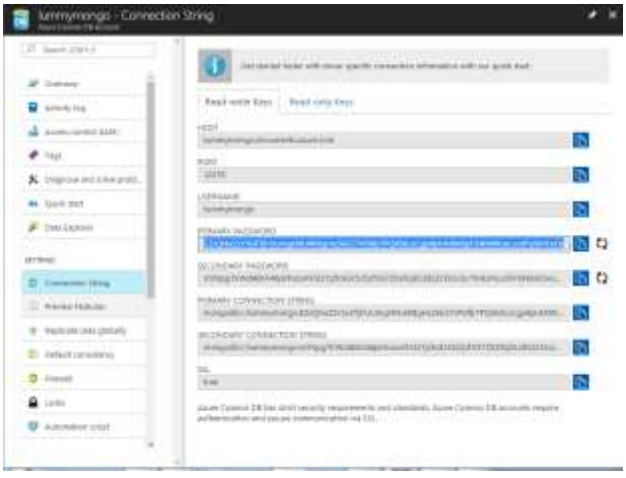
```
{
    "$schema": "http://schema.management.azure.com/schemas/2014-04-01-preview/deploymentTemplate.json#",
    "contentVersion": "1.0.0.0",
    "parameters": {
        "name": {
            "type": "string"
        },
        "location": {
            "type": "string"
        },
        "locationName": {
            "type": "string"
        },
        "defaultExperience": {
            "type": "string"
        },
        "pairedLocation": {
            "type": "string"
        },
        "pairedLocationName": {
            "type": "string"
        }
    },
    "resources": [
        {
            "apiVersion": "2015-04-08",
            "kind": "MongoDB",
            "type": "Microsoft.DocumentDb/databaseAccounts",
            "name": "[parameters('name')]",
            "location": "[parameters('location')]",
            "properties": {
                "databaseAccountOfferType": "Standard",
                "locations": [
                    {
                        "id": "[concat(parameters('name'), '-', parameters('location'))]",
                        "failoverPriority": 0,
                        "locationName": "[parameters('locationName')]"
                    }
                ]
            },
            "tags": {
                "defaultExperience": "[parameters('defaultExperience')]"
            }
        }
    ]
}
```

Deploy HD Insight and MongoDB from Azure CLI

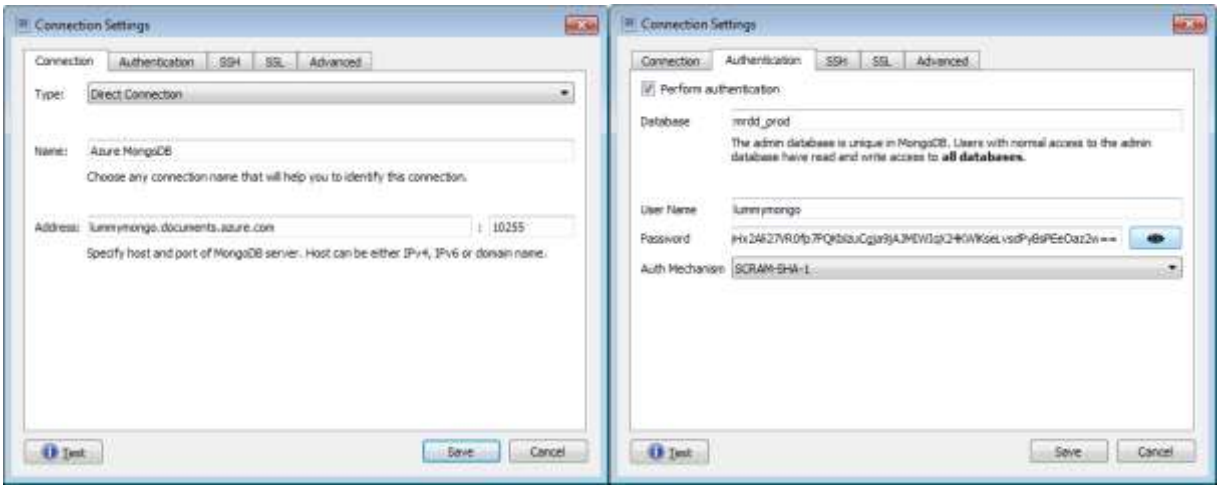
```
Microsoft Windows [Version 6.1.7601]
Copyright (c) 2009 Microsoft Corporation.  All rights reserved.

C:\Users\ekgriol>cd \DeepAzure_FinalProject

C:\DeepAzure_FinalProject>az group deployment create --name lummyhd ^
More? --resource-group lummyrg ^
More? --template-file lummyhd-template.json ^
More? --parameters lummyhd-parameters.json ^
More? --output table
Name     ResourceGroup
-------  ---------------
lummyhd  lummyrg

C:\DeepAzure_FinalProject>
```

```
Microsoft Windows [Version 6.1.7601]
Copyright (c) 2009 Microsoft Corporation.  All rights reserved.

C:\Users\ekgriol>cd \DeepAzure_FinalProject

C:\DeepAzure_FinalProject>az group deployment create --name lummymongo ^
More? --resource-group lummyrg ^
More? --template-file lummymongo-template.json ^
More? --parameters lummymongo-parameters.json ^
More? --output table
Name         ResourceGroup
-----------  ---------------
lummymongo  lummyrg

C:\DeepAzure_FinalProject>
```

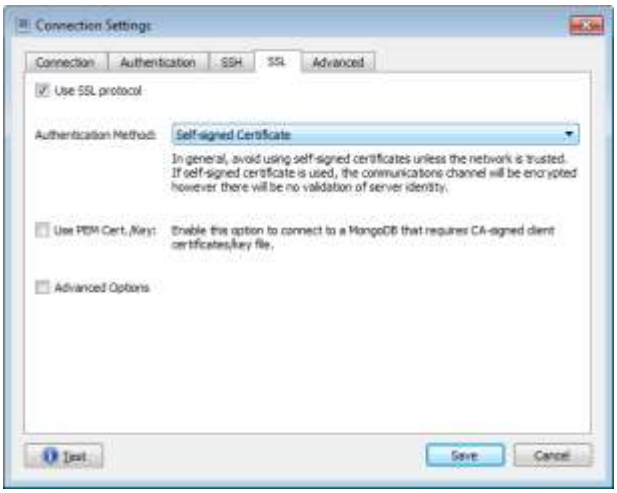
Once deployed, open up the LummyMongo Connection String blade and copy out the Primary Password



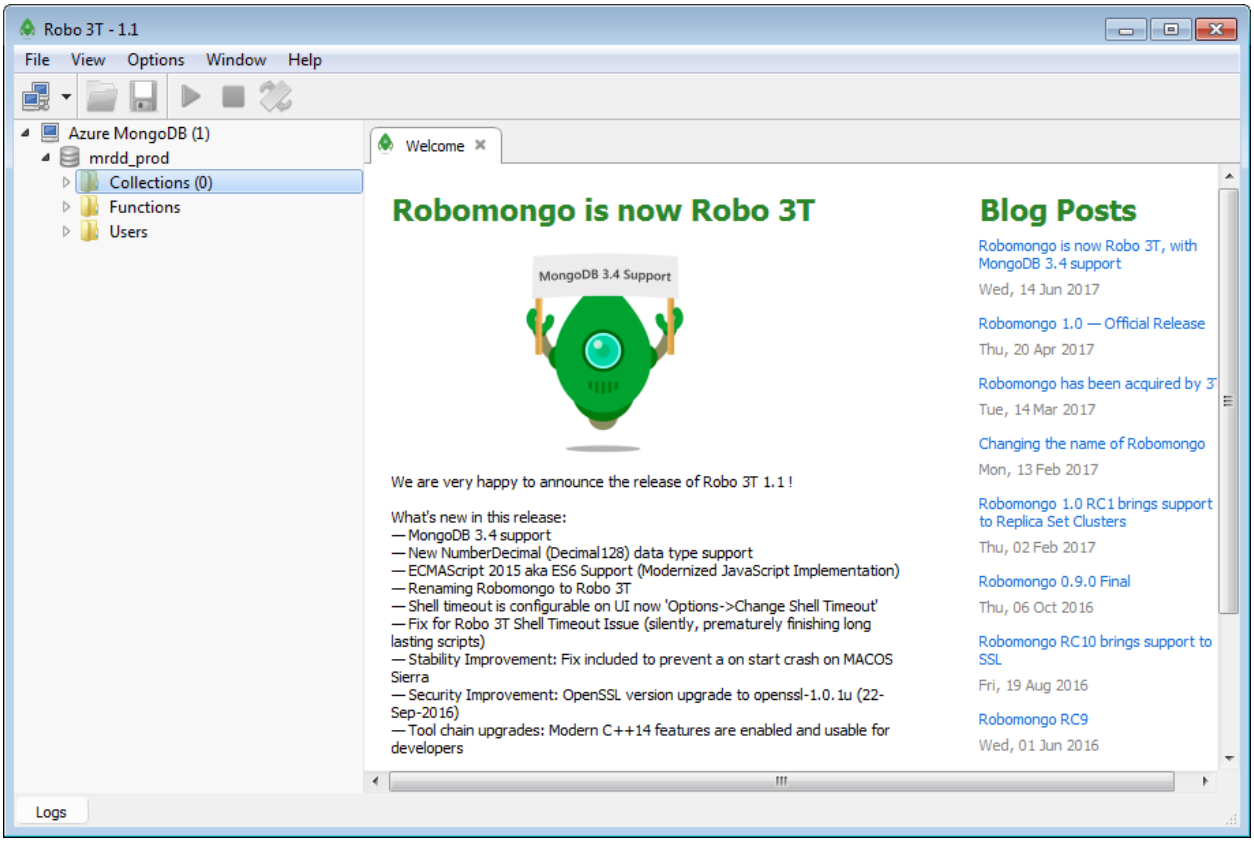
Paste the primary password into RoboMongo 3T. Use lummymongo.documents.azure.com:10255 as the host:port and use SSL.



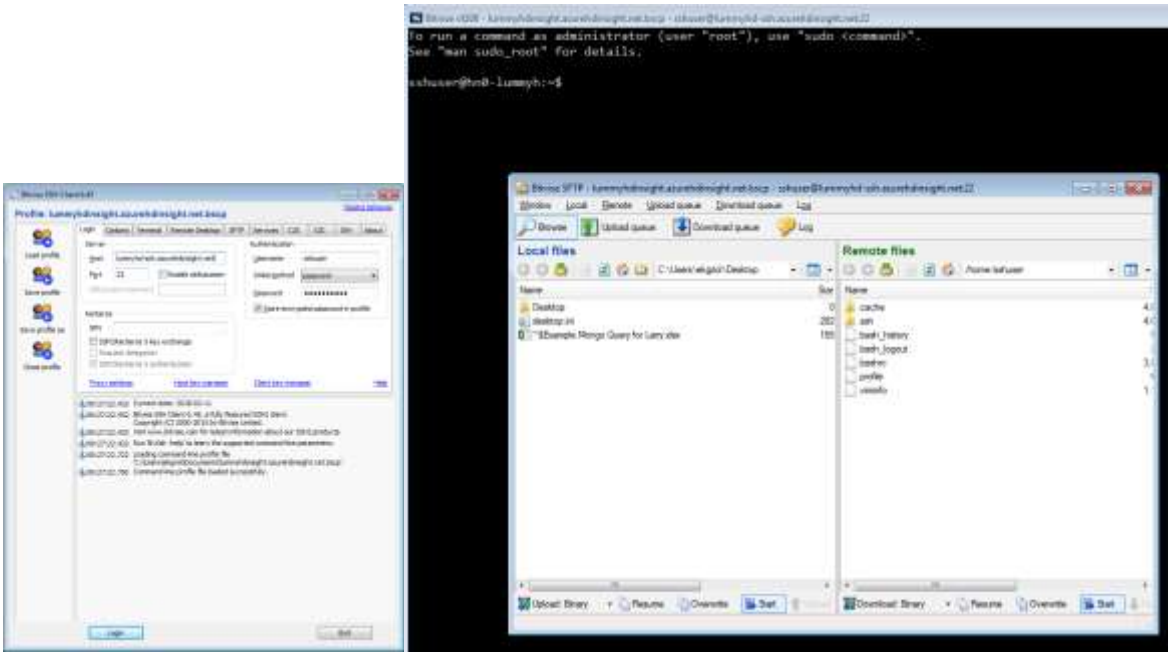
Set “Use SSL Protocol” and “Self-Signed Certificate” to allow a remote connection into CosmosDB.



Now I save the connection and log in, verifying connectivity.

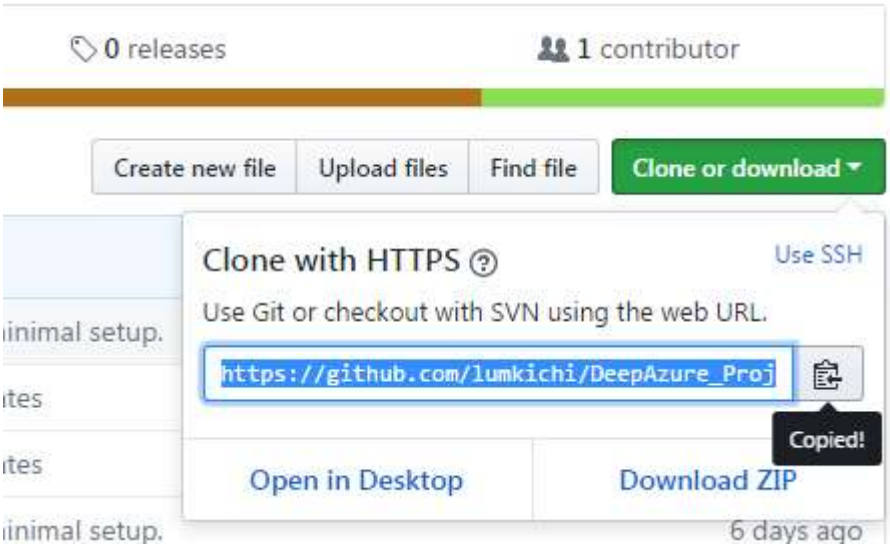


Connect to the HD Insight Node using the Hostname, User and Password setup in the template/parameters file, and click login



We should be greeted with a SSH shell prompt and a SFTP window ready-to-go.

From the https://github.com/lumkichi/DeepAzure_Project page, click the green CLONE OR DOWNLOAD button and copy out the URL.

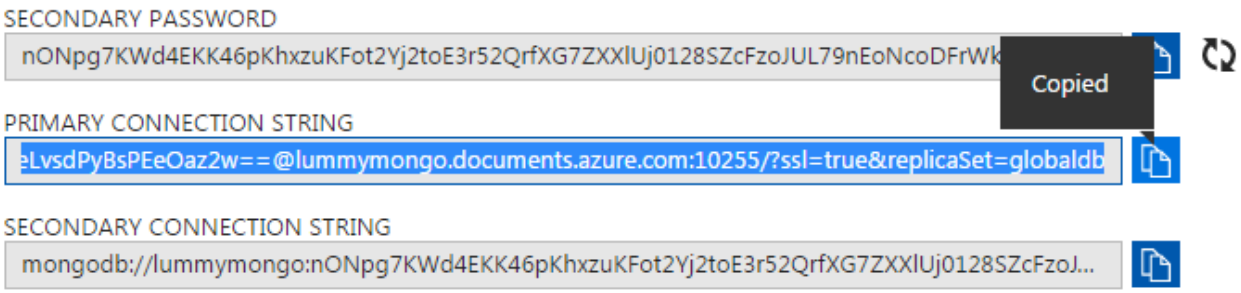


From the Headnode SSH prompt, clone the git repository into the local user's homedir

```
To run a command as administrator (user "root"), use "sudo <command>".
See "man sudo_root" for details.

sshuser@hn0-lummyh:~$ git clone https://github.com/lumkichi/DeepAzure_Project.git
Cloning into 'DeepAzure_Project'...
remote: Counting objects: 40, done.
remote: Compressing objects: 100% (30/30), done.
remote: Total 40 (delta 5), reused 37 (delta 4), pack-reused 0
Unpacking objects: 100% (40/40), done.
Checking connectivity... done.
sshuser@hn0-lummyh:~$ ls -l
total 4
drwxrwxr-x 9 sshuser sshuser 4096 Feb 11 05:31 DeepAzure_Project
sshuser@hn0-lummyh:~$ cd DeepAzure_Project/
sshuser@hn0-lummyh:~/DeepAzure_Project$ ls -l
total 36
drwxrwxr-x 2 sshuser sshuser 4096 Feb 11 05:31 build
drwxrwxr-x 2 sshuser sshuser 4096 Feb 11 05:31 creation_templates
drwxrwxr-x 2 sshuser sshuser 4096 Feb 11 05:31 data
drwxrwxr-x 2 sshuser sshuser 4096 Feb 11 05:31 dist
drwxrwxr-x 2 sshuser sshuser 4096 Feb 11 05:31 lib
-rwxrwxr-x 1 sshuser sshuser 2059 Feb 11 05:31 mongo_compile.sh
-rwxrwxr-x 1 sshuser sshuser 1841 Feb 11 05:31 mongo_run.sh
-rw-rw-r-- 1 sshuser sshuser  280 Feb 11 05:31 README.md
drwxrwxr-x 3 sshuser sshuser 4096 Feb 11 05:31 src
sshuser@hn0-lummyh:~/DeepAzure_Project$ cd data
sshuser@hn0-lummyh:~/DeepAzure_Project/data$ ls -l
total 4884
-rw-rw-r-- 1 sshuser sshuser 4997259 Feb 11 05:31 RXBC_20180122_INSERT_EXTRACT.dat.gz
sshuser@hn0-lummyh:~/DeepAzure_Project/data$
```

A 42MB Datafile (compressed to 4.9 MB) would have been copied into the ~/data folder.

From the LummyMongo Connection String blade, copy out the PRIMARY CONNECTION STRING



We would need to edit a local file, “MongoConnection.java” to include this connection string info, and recompile.

```
sshuser@hn0-lummyh:~/DeepAzure_Project$ vi src/com/relayhealth/mongoloader/util/MongoConnection.java
```

Replace the highlighted string with the PRIMARY CONNECTION STRING copied from Azure Portal and save the file.

```
*/
public class MongoConnection {

  private MongoClient mongoClient = null;
  private MongoDatabase mongoDB = null;

  public MongoConnection() {

    try {
      // Copy out the MongoClient string from the Azure Portal
      mongoClient = new MongoClient(new
MongoClientURI("mongodb://lummymongo:Hb7oqBPsctlYiWq7qljdHg9sAIdDCOVladpTsRab32GdjluEJaY45ZtKta5Y3Vsxnt
UhU0afpE41kIxezeoVmA==@lummymongo.documents.azure.com:10255/?ssl=true&replicaSet=globaldb"));
      mongoDB = mongoClient.getDatabase("mrdd_prod");
    } catch (Exception e) {
      e.printStackTrace();
      System.exit(1);
    }
  }

  public MongoCollection<Document> getMRDDCollection() {
    return mongoDB.getCollection("MRDD");
  }
```

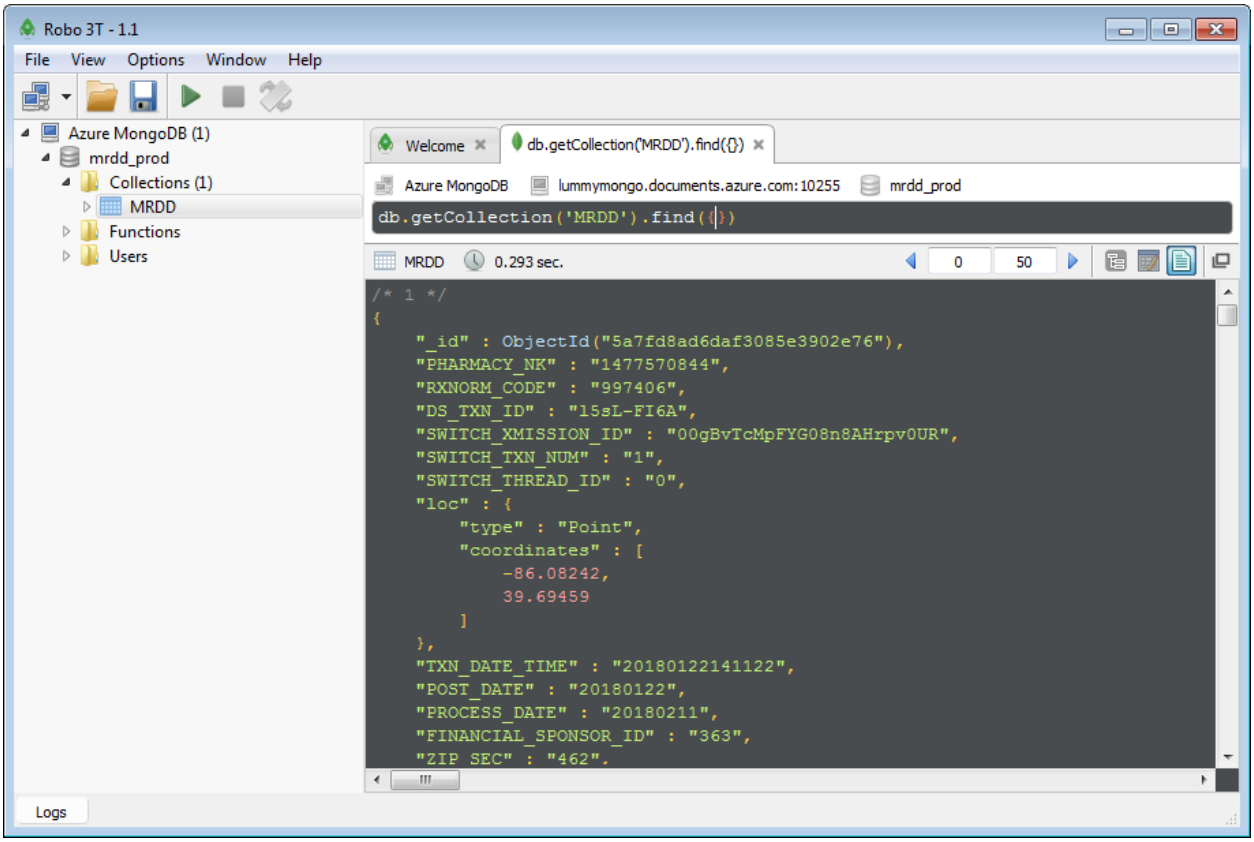
The run “mongo_compile.sh” which will recompile all the java classes, unpack the library files (mongo java driver, and slf4j nop and build a complete executable jar file in the dist/ folder.

```
"src/com/relayhealth/mongoloader/util/MongoConnection.java" 52L, 1367C written
sshuser@hn0-lummyh:~/DeepAzure_Project$ ls -ltr
total 36
-rw-rw-r-- 1 sshuser sshuser  280 Feb 11 05:31 README.md
drwxrwxr-x 2 sshuser sshuser 4096 Feb 11 05:31 data
drwxrwxr-x 2 sshuser sshuser 4096 Feb 11 05:31 creation_templates
drwxrwxr-x 2 sshuser sshuser 4096 Feb 11 05:31 build
drwxrwxr-x 2 sshuser sshuser 4096 Feb 11 05:31 dist
drwxrwxr-x 3 sshuser sshuser 4096 Feb 11 05:31 src
-rwxrwxr-x 1 sshuser sshuser 1841 Feb 11 05:31 mongo_run.sh
-rwxrwxr-x 1 sshuser sshuser 2059 Feb 11 05:31 mongo_compile.sh
drwxrwxr-x 2 sshuser sshuser 4096 Feb 11 05:31 lib
sshuser@hn0-lummyh:~/DeepAzure_Project$ ./mongo_compile.sh
Setting env variables
Setting output dirs

Compiling
  MongoMap.java
Note: com/relayhealth/hadoop/util/MongoMap.java uses or overrides a deprecated API.
Note: Recompile with -Xlint:deprecation for details.
  MongoReduce.java
  MongoConnection.java
  MongoLoad.java
Note: ./com/relayhealth/hadoop/util/MongoMap.java uses or overrides a deprecated API.
Note: Recompile with -Xlint:deprecation for details.

Building jar
Jar file at ~/DeepAzure_Project/dist/MongoLoad.jar

Complete
sshuser@hn0-lummyh:~/DeepAzure_Project$
```

Now we are ready to simply kick off the run. The 'mongo_run.sh' will unpack the gzipped data file if necessary, create the HDFS folders (/user/hadoop/mongo_input/) and copy the data file to it, then kick off the MapReduce program "MongoLoad.jar"

```
sshuser@hn0-lummyh:~/DeepAzure_Project$ ./mongo_run.sh
Setting env variables
rmr: DEPRECATED: Please use 'rm -r' instead.
rmr: `/user/hadoop/mongo_output': No such file or directory
SLF4J: Failed to load class "org.slf4j.impl.StaticLoggerBinder".
SLF4J: Defaulting to no-operation (NOP) logger implementation
SLF4J: See http://www.slf4j.org/codes.html#StaticLoggerBinder for further details.
18/02/11 05:45:59 INFO client.AHSProxy: Connecting to Application History server at headnodehost/10.0.0.19:10200
18/02/11 05:45:59 INFO client.RequestHedgingRMFailoverProxyProvider: Looking for the active RM in [rm1, rm2]...
18/02/11 05:46:00 INFO client.RequestHedgingRMFailoverProxyProvider: Found active RM [rm1]
18/02/11 05:46:00 INFO input.FileInputFormat: Total input paths to process : 1
18/02/11 05:46:00 INFO mapreduce.JobSubmitter: number of splits:3
18/02/11 05:46:00 INFO mapreduce.JobSubmitter: Submitting tokens for job: job_1518238224323_0002
18/02/11 05:46:01 INFO impl.YarnClientImpl: Submitted application application_1518238224323_0002
18/02/11 05:46:01 INFO mapreduce.Job: The url to track the job: http://hn0-
lummyh.kuyd0rytcc4uj12gzzawh350og.bx.internal.cloudapp.net:8088/proxy/application_1518238224323_0002/
18/02/11 05:46:01 INFO mapreduce.Job: Running job: job_1518238224323_0002
18/02/11 05:46:17 INFO mapreduce.Job: Job job_1518238224323_0002 running in uber mode : false
18/02/11 05:46:17 INFO mapreduce.Job:  map 0% reduce 0%
18/02/11 05:46:34 INFO mapreduce.Job:  map 1% reduce 0%
18/02/11 05:46:55 INFO mapreduce.Job:  map 2% reduce 0%
18/02/11 05:47:14 INFO mapreduce.Job:  map 3% reduce 0%
```

While this is running, we refresh RoboMongo 3T and see that a new MRDD collection has been created and querying it, data is present:



The data file has been obtained from the Data Services grid, as part of a daily QA Test Flat File based on real-world data and has the following format:

Field Name	Source Data Type
Action	CHAR(1) - 'A'/'D'/'U'
PHARMACY_NK	VARCHAR2(10) – NPI
RXNORM_CODE	VARCHAR2(15) – RxNorm
DS_TXN_ID	VARCHAR2(14)
SWITCH_XMISSION_ID	VARCHAR2(40)
SWITCH_TXN_ID	INT
SWITCH_THREAD_ID	INT
NCPDP_LOC_LAT	VARCHAR2(50)
NCPDP_LOC_LONG	VARCHAR2(50)
TXN_DATE_TIME	DATE
POST_DATE	CHAR(8) – YYYYMMDD
FINANCIAL_SPONSOR_ID	VARCHAR2(10)
ZIP_SEC	VARCHAR2(3)
NCPDP_LOC_STATE	VARCHAR2(2)
REQUEST_B1_B3	VARCHAR2(4000)

MongoLoader.java is the main class which is called by the Hadoop framework to setup the MapReduce job. It is responsible for defining the Map and Reduce classes to be used, the define the input and output folders to be used. Additionally, in this installation – we can configure the splitting of the incoming datafile and also to set the job.setNumReduceTasks() to zero.

```
package com.relayhealth.mongoloader;

import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapred.JobConf;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
import org.apache.hadoop.util.GenericOptionsParser;

import com.relayhealth.hadoop.util.MongoMap;
import com.relayhealth.hadoop.util.MongoReduce;

public class MongoLoad {

  final static long DEFAULT_SPLIT_SIZE = 128 * 1024 * 1024;

  public static void main(String[] args) throws Exception {
    JobConf conf = new JobConf();

    // The following line can influence the number of splits in the file.  The higher the divisor, the more split
    // files.  Currently we utilize the default as CosmosDB itself does not handle high volume rate at its
    // current configuration (thus would return an error).
    conf.setLong(FileInputFormat.SPLIT_MAXSIZE, conf.getLong(FileInputFormat.SPLIT_MAXSIZE, DEFAULT_SPLIT_SIZE) / 4);

    String[] otherArgs = new GenericOptionsParser(conf, args).getRemainingArgs();
    if (otherArgs.length != 2) {
      System.err.println("Usage: <in> <out>");
      System.exit(2);
    }

    // Setup the Job instance, and provide a name so its progress can be tracked.
    Job job = Job.getInstance(conf, "MongoLoad");

    // Set the Mapper and Reducer classes
    job.setMapperClass(MongoMap.class);
    job.setReducerClass(MongoReduce.class);

    // Provide the main class for this jar
    job.setJarByClass(MongoLoad.class);

    // Set the input/output key/value pairs to Text
    job.setMapOutputKeyClass(Text.class);
    job.setMapOutputValueClass(Text.class);
    job.setOutputKeyClass(Text.class);
    job.setOutputValueClass(Text.class);

    // Turn off the reducer tasks - we do not need it here
    job.setNumReduceTasks(0);

    // Define the input file-path
    FileInputFormat.addInputPath(job, new Path(otherArgs[0]));

    // Define the output file-path (so that temporary and success files can be written there)
    FileOutputFormat.setOutputPath(job, new Path(otherArgs[1]));

    // Launch the job and wait for completion.
    System.exit(job.waitForCompletion(true) ? 0 : 1);
  }
}
```

Once the job is configured, the Hadoop framework splits the file and marshals it along with the code to be run on the remote servers. It first calls the map method in the MongoMap.java. This is the workhorse of the MongoLoad process; it receives one line of text, parses it, builds the Mongo bson object and performs the requested task (Insert / Update / Delete). When the MongoMap object is instantiated, it makes a connection to the MongoDB using the MongoConnection class.

```
package com.relayhealth.hadoop.util;


import java.io.IOException;
import java.sql.Timestamp;
import java.util.ArrayList;
import java.util.List;

import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Mapper;
import org.bson.Document;

import com.mongodb.BasicDBObject;
import com.mongodb.DBObject;
import com.mongodb.client.MongoCollection;
import com.mongodb.client.result.UpdateResult;
import com.mongodb.util.JSON;
import com.relayhealth.mongoloader.util.MongoConnection;

/**
 * The Mongo DB data load worker extending the Mapper object, called by the Map/Reduce
 * infrastructure.  Parses one line provided by Map/Reduce and connects (if necessary)
 * to MongoDB and inserts/updates/deletes a document (record).
 * @author lawrence.spiwak@relayhealth.com
 *
 */
/**
 * @author ekgriol
 *
 */
@SuppressWarnings("deprecation")
public class MongoMap extends Mapper<Object, Text, Text, Text>  {

  boolean dirtyInsert = false;
  String content = "";
  String processDate = "";
  char sepChar = 02;

  // Mongo Connection
  MongoConnection mongoConn = null;
  MongoCollection<Document> collection = null;


  // Various field ID's to load into Mongo
  private static String PHARMACY_NK = "PHARMACY_NK";
  private static String RXNORM_CODE = "RXNORM_CODE";
  private static String DS_TXN_ID = "DS_TXN_ID";
  private static String SWITCH_XMISSION_ID = "SWITCH_XMISSION_ID";
  private static String SWITCH_TXN_NUM = "SWITCH_TXN_NUM";
  private static String SWITCH_THREAD_ID = "SWITCH_THREAD_ID";
  private static String LOC = "loc";
  private static String REQUEST_B1B3 = "REQUEST_B1B3";
  private static String FINANCIAL_SPONSOR_ID = "FINANCIAL_SPONSOR_ID";
  private static String ZIP_SEC = "ZIP_SEC";
  private static String NCPDP_LOC_STATE = "STATE";
  private static String TXN_DATE_TIME = "TXN_DATE_TIME";
  private static String POST_DATE = "POST_DATE";
  private static String PROCESS_DATE = "PROCESS_DATE";

  public MongoMap() {
    // On creation of object, make a connection to MongoDB
    if (mongoConn == null) {
      mongoConn = new MongoConnection();
      collection = mongoConn.getMRDDCollection();
    }

    // Get the process date as today's date YYYYMMDD
    java.util.Date date= new java.util.Date();
```

```
    Timestamp ts = new Timestamp(date.getTime());
    processDate = ts.toString().substring(0,10).replace("-","");
  }

  public void finalize() throws Throwable {
    // On deletion of object, clean up connection to MongoDB
    if (mongoConn != null) {
      mongoConn.close();
    }

    super.finalize();
  }

  /**
   * Map/Reduce will call this map() method passing in the object Key, Value and Context<br/>
   * We are concerned mainly with the 'value' parameter.  This class has been simplified from
   * the original MRDDLoader which supported bulk inserts - Map/Reduce feeds this class one
   * line at a time.
   */
  public void map(Object key, Text value, Context context) throws IOException, InterruptedException {

    // Split the incoming line (delimited by 0x02)
    String[] tokens = value.toString().split("" + sepChar);

    // Check line format for correct number of columns
    if (tokens.length != 12 && tokens.length != 15 && tokens.length != 3) {
      System.err.println("Line does not have the expected values of 3 (del), 12 (old-style ins/upd), 15 (new-style
ins/upd) fields");
      System.err.println("Line = " + content);
      System.exit(1);
    }

    // tokens[0] contains the "A" add "U" update "D" delete action flag.  Uppercase it, if non-null
    if (tokens[0] != null) {
      tokens[0] = tokens[0].toUpperCase();
    }


    // Validate the Action token
    if ("A".equals(tokens[0]) || "U".equals(tokens[0]) || "D".equals(tokens[0])) {

      // In this block we are handling Add records
      if ("A".equals(tokens[0])) {
        if (tokens.length != 15 && tokens.length != 12) {
          System.err.println("Received an ADD record without 12 or 15 columns (" + tokens.length + " columns)");
          System.exit(1);
        }

        // A list of doc for bulk insert (not needed here)
        List<Document> insertDocList = new ArrayList<Document>();

        // Build a new bson/json document
        Document doc = new Document();
        doc.append(PHARMACY_NK,tokens[1]);
        doc.append(RXNORM_CODE,tokens[2]);
        doc.append(DS_TXN_ID,tokens[3]);
        doc.append(SWITCH_XMISSION_ID,tokens[4]);
        doc.append(SWITCH_TXN_NUM,tokens[5]);
        doc.append(SWITCH_THREAD_ID,tokens[6]);

        // If lat or long is empty, do not insert a coordinate object
        if (tokens[7] != null && tokens[8] != null && !"".equals(tokens[7].trim()) && !"".equals(tokens[8].trim())) {
          // GeoJSON Point Objects are [longitude, latitude] and not [latitude, longitude]
          String coordinatesJSON = "{ type: \"Point\", coordinates: [ " +  tokens[8] + ", " + tokens[7] + " ] }";
          try {
            DBObject coordinatesObj = (DBObject) JSON.parse(coordinatesJSON);
            doc.append(LOC, coordinatesObj);
          } catch (Exception e) {}
        } else {
          // System.out.println("Received a line with no Lat/Long - not creating a coordinate object: DS_TXN_ID = " +
tokens[3]);
        }
        doc.append(TXN_DATE_TIME, tokens[9]);
        doc.append(POST_DATE, tokens[10]);
        doc.append(PROCESS_DATE, processDate);
        if (tokens.length == 12) {
          doc.append(REQUEST_B1B3,tokens[11].getBytes());
        } else {
```

```
          doc.append(FINANCIAL_SPONSOR_ID, tokens[11]);
          doc.append(ZIP_SEC, tokens[12]);
          doc.append(NCPDP_LOC_STATE, tokens[13]);
          doc.append(REQUEST_B1B3,tokens[14].getBytes());
        }

        // Add this record to the bulk list (vestigial)
        insertDocList.add(doc);
        dirtyInsert = true;

        // Push data to mongo every bulkSize records (one record, in this case)
        try {
          collection.insertMany(insertDocList);
        } catch (Exception e) {
          // Caught a duplicate key exception, most likely - reattempt to iterate through the bulk and insert
          // individual rows - if duplicate key, reattempt using updates
          System.out.println("Bulk insert failed - attempting single insert/update");
          for (Document obj: insertDocList) {
            try {
              // Attempt an single document insert.
              collection.insertOne(obj);

            } catch (Exception e1) {
              // If insert fails, attempt update
              String pharmacy_nk = obj.getString(PHARMACY_NK);
              String rxnorm_code = obj.getString(RXNORM_CODE);

              BasicDBObject searchQuery = new BasicDBObject();
              searchQuery.append(PHARMACY_NK,pharmacy_nk);
              searchQuery.append(RXNORM_CODE,rxnorm_code);
              BasicDBObject newDoc = new BasicDBObject().append("$set", obj);

              try {
                collection.updateOne(searchQuery, newDoc);
              } catch (Exception e2) {
                System.err.println("Caught an Exception trying to update to DB");
                System.err.println("Key was PHARMACY_NK=" + pharmacy_nk + "  RXNORM_CODE=" + rxnorm_code);
                e.printStackTrace();
                System.exit(1);
              }

            }
          }
        }

        dirtyInsert = false;
        insertDocList = new ArrayList<Document>();

        // In this block we handle Update records.  Attempt to update, if it fails attempt an insert.
      } else if ("U".equals(tokens[0])) {
        if (tokens.length != 15 && tokens.length != 12) {
          System.err.println("Received an UPDATE record without 12 or 15 columns (" + tokens.length + " columns)");
          System.exit(1);
        }

        BasicDBObject searchQuery = new BasicDBObject();
        searchQuery.append(PHARMACY_NK,tokens[1]);
        searchQuery.append(RXNORM_CODE,tokens[2]);
        Document updateDoc = new Document();
        updateDoc.append(PHARMACY_NK,tokens[1]);
        updateDoc.append(RXNORM_CODE,tokens[2]);
        updateDoc.append(DS_TXN_ID,tokens[3]);
        updateDoc.append(SWITCH_XMISSION_ID,tokens[4]);
        updateDoc.append(SWITCH_TXN_NUM,tokens[5]);
        updateDoc.append(SWITCH_THREAD_ID,tokens[6]);

        // If lat or long is empty, do not insert a coordinate object
        if (tokens[7] != null && tokens[8] != null && !"".equals(tokens[7].trim()) && !"".equals(tokens[8].trim())) {
          // GeoJSON Point Objects are [longitude, latitude] and not [latitude, longitude]
          String coordinatesJSON = "{ type: \"Point\", coordinates: [ " +  tokens[8] + ", " + tokens[7] + " ] }";
          try {
            DBObject coordinatesObj = (DBObject) JSON.parse(coordinatesJSON);
            updateDoc.append(LOC, coordinatesObj);
          } catch (Exception e) {
          }
        }else {
          System.out.println("Received a line with no Lat/Long - not creating a coordinate object: DS_TXN_ID = " +
tokens[3]);
```

```
        }

        updateDoc.append(TXN_DATE_TIME, tokens[9]);
        updateDoc.append(POST_DATE, tokens[10]);
        updateDoc.append(PROCESS_DATE, processDate);
        if (tokens.length == 12) {
          updateDoc.append(REQUEST_B1B3,tokens[11].getBytes());
        } else {
          updateDoc.append(FINANCIAL_SPONSOR_ID, tokens[11]);
          updateDoc.append(ZIP_SEC, tokens[12]);
          updateDoc.append(NCPDP_LOC_STATE, tokens[13]);
          updateDoc.append(REQUEST_B1B3,tokens[14].getBytes());
        }
        BasicDBObject newDoc = new BasicDBObject().append("$set", updateDoc);

        try {
          UpdateResult ur = collection.updateOne(searchQuery, newDoc);

          // An update failed to update an existing record - so try an insert instead
          if (ur.getModifiedCount() == 0) {
            try {
              collection.insertOne(updateDoc);
            } catch (Exception e) {
              System.out.println("Update : PHARMACY_NK = " + tokens[1] + "  RNXORM_CODE = " + tokens[2] + " - failed
to update and insert.  Exception was " + e.getMessage());
            }
          }
        } catch (Exception e) {
          e.printStackTrace();
          System.exit(1);
        }

        // Here we handled Delete records.  Simply delete the referenced document by its key
      } else {
        // Do the deletion
        BasicDBObject searchQuery = new BasicDBObject();
        searchQuery.append(PHARMACY_NK, tokens[1]);
        searchQuery.append(RXNORM_CODE,tokens[2]);
        try {
          collection.deleteMany(searchQuery);
        } catch (Exception e) {
          e.printStackTrace();
          System.exit(1);
        }
      }
    } else {
      System.err.println("Line does not have the expected Action of 'A', 'U' or 'D' - found '" + tokens[0] + "'
instead.");
      System.err.println("Line = " + content);
      System.exit(1);
    }
  }

}
```

The MongoReduce.java is a No-Op class, provided for completeness. It is a simple, empty class which does nothing. Once the data has been loaded into Mongo, there is no step required at this point to consolidate any data for post-processing.

```
package com.relayhealth.hadoop.util;

import java.io.IOException;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Reducer;

/**
 * A NoOp Class which extends the Reducer object but does no actual work
 *
 * @author lawrence.spiwak@relayhealth.com
 *
 */
public class MongoReduce extends Reducer<Text, Text, Text, Text> {
         public void reduce(Text key, Iterable<Text> values, Context context) throws IOException, InterruptedException
{
         }
}
```

Lastly, the MongoConnection.java is the utility class that connects to MongoDB, and passes a collections object back to MongoMap that can be used. *NOTE* this class can and should be extended to be a static singleton class, with provisions for connection pooling to improve the connection performance. But for this project, I kept it simple. The part highlighted in yellow should be replaced with the PRIMARY CONNECTION STRING copied from the Azure Portal.

```
package com.relayhealth.mongoloader.util;

import org.bson.Document;

import com.mongodb.MongoClientURI;
import com.mongodb.MongoClient;
import com.mongodb.client.MongoCollection;
import com.mongodb.client.MongoDatabase;

/**
 * A utility class simplified for making direct connections to MongoDB in
 * Microsoft Azure's Cosmos DB
 * @author lawrence.spiwak@relayhealth.com
 *
 */
public class MongoConnection {

  private MongoClient mongoClient = null;
  private MongoDatabase mongoDB = null;

  public MongoConnection() {

    try {
      mongoClient = new MongoClient(new
MongoClientURI("mongodb://lummymongo:Hb7oqBPsctlYiWq7qljdHg9sAIdDCOVladpTsRab32GdjluEJaY45ZtKta5Y3VsxntUhU0afpE41kIxez
eoVmA==@lummymongo.documents.azure.com:10255/?ssl=true&replicaSet=globaldb"));
      mongoDB = mongoClient.getDatabase("mrdd_prod");
    } catch (Exception e) {
      e.printStackTrace();
      System.exit(1);
    }
  }

  public MongoCollection<Document> getMRDDCollection() {
    return mongoDB.getCollection("MRDD");
  }

  public MongoCollection<Document> getPBICollection() {
    return mongoDB.getCollection("PBI");
  }
```

```
  public MongoDatabase getMongoDB() {
    return mongoDB;
  }

  public void close() {
    if (mongoClient != null) {
      mongoClient.close();
    }
    mongoClient = null;
  }

}
```

The 'mongo_compile.sh' script compiles each of the Java code using the Hadoop extended compiler. It sets the project path, checks for the presence of Java 8 and adds it to the beginning of the PATH variable, sets the HADOOP_CLASSPATH variable. It also enumerates all of the java library files in the ./lib folder and adds them to the HADOOP_CLASSPATH.

Then it creates the output folders ./build and ./dist folders, removes and previously compiled *.class files and begins compiling the code, stopping at any point if there is a compile error that needs to be resolved. We will be using the "/usr/bin/hadoop com.sun.tools.javac" compiler and not the JDK8 javac compiler.

The compiled binaries get delivered to the ./build folder. We also unpack each of the library files (mongo driver and slf4j nop) into the ./build folder as well, renaming the respective MANIFEST.MF files to preserve their license information. We rebuild a brand new jar with our own MANIFEST.MF and drop it into the ./dist folder.

```
#!/bin/bash

# Set the Root Project Folder
ROOT=DeepAzure_Project

# Check if the PATH contains java
echo "Setting env variables"
java_setup=`echo $PATH | grep 'java-8'`
if [ -z "$java_setup" ]
then
  export PATH=${JAVA_HOME}/bin:$PATH
fi

# Check if HADOOP_CLASSPATH exists, otherwise set it up
if [ -z "${HADOOP_CLASSPATH}" ]
then
  export HADOOP_CLASSPATH=.:${JAVA_HOME}/lib/tools.jar
fi

# Iterate all jar files in lib folder and add to classpath
for II in `ls ~/$ROOT/lib`
do
  CP=$CP:~/$ROOT/lib/$II
done
export HADOOP_CLASSPATH=$HADOOP_CLASSPATH$CP


# Create the output dirs
echo "Setting output dirs"
mkdir -p ~/$ROOT/build
mkdir -p ~/$ROOT/dist
rm -f ~/$ROOT/build/*.class

# Compile the individual classes
echo " "
echo "Compiling"
cd ~/$ROOT/src
```

```
echo "  MongoMap.java"
/usr/bin/hadoop com.sun.tools.javac.Main -d ~/$ROOT/build  com/relayhealth/hadoop/util/MongoMap.java
if [ $? != 0 ]
then
  exit
fi

echo "  MongoReduce.java"
/usr/bin/hadoop com.sun.tools.javac.Main -d ~/$ROOT/build  com/relayhealth/hadoop/util/MongoReduce.java
if [ $? != 0 ]
then
  exit
fi

echo "  MongoConnection.java"
/usr/bin/hadoop com.sun.tools.javac.Main -d ~/$ROOT/build  com/relayhealth/mongoloader/util/MongoConnection.java
if [ $? != 0 ]
then
  exit
fi

echo "  MongoLoad.java"
/usr/bin/hadoop com.sun.tools.javac.Main -d ~/$ROOT/build  com/relayhealth/mongoloader/MongoLoad.java
if [ $? != 0 ]
then
  exit
fi

# Create the jar file
echo " "
echo "Building jar"
cd ~/$ROOT/build

# Unpack the contents of the Mongo driver and the No OP slf4j libraries
# Rename the respective MANIFEST.MF's to preserve the licensing info
unzip -qq -o ../lib/mongo*.jar
mv META-INF/MANIFEST.MF META-INF/MANIFEST-SLF4J.MF
unzip -qq -o ../lib/slf4j*.jar
mv META-INF/MANIFEST.MF META-INF/MANIFEST-MONGO.MF

# Assert our own MANIFEST.MF into the target Jar and build the new jar
# which includes all of the Mongo Driver and SLF4J libraries as well.
jar cfm ~/$ROOT/dist/MongoLoad.jar ~/$ROOT/dist/MANIFEST.MF ./*
echo "Jar file at ~/$ROOT/dist/MongoLoad.jar"
echo " "
echo "Complete"
```

Lastly, the mongo_run.sh sets up the environment to run the MongoLoad.jar that was created above. It builds the classpath variables similarly as the mongo_compile. One thing it *does* do here is to rename the slf4j-log4j api jar to something else so that the Mongo Driver doesn't belch out INFO and DEBUG statements to the console, and so that it won't collide with the slf4j nop jar that I've provided to suppress the output. Once the run is complete, the original jar file is restored.

The script goes on to unpack the gzipped file, if present, load the uncompressed datafile into the HDFS under /user/hadoop/mongo_input folder. It also removes any mongo_output folder that might be present from a previous run. The MongoLoad.jar is kicked off as follows:

```
hadoop jar ~/$ROOT/dist/MongoLoad.jar /user/hadoop/mongo_input/RXBC_20180122_INSERT_EXTRACT.dat /user/hadoop/mongo_output
```

```
#!/bin/bash

# Set the Root Project Folder
ROOT=DeepAzure_Project

# Check if the PATH contains java
echo "Setting env variables"
java_setup=`echo $PATH | grep 'java-8'`
if [ -z "$java_setup" ]
then
  export PATH=${JAVA_HOME}/bin:$PATH
fi

# Check if HADOOP_CLASSPATH exists, otherwise set it up
if [ -z "${HADOOP_CLASSPATH}" ]
then
  export HADOOP_CLASSPATH=.:${JAVA_HOME}/lib/tools.jar
fi

# Restore any SLF4J jars that has been disabled by this script.  SLF4J causes
# excessive logging in the Mongo Driver, so it has to be disabled for the
# duration of the run.
sudo mv /usr/hdp/2.6.2.3-1/hadoop/lib/slf4j-log4j12-1.7.10.jarx /usr/hdp/2.6.2.3-1/hadoop/lib/slf4j-log4j12-1.7.10.jar
2>/dev/null

# If the data file exists in the $ROOT/data folder as a gzipped file,
# unzip it before proceeding
if [ -e ~/$ROOT/data/RXBC_20180122_INSERT_EXTRACT.dat.gz ]
then
  cd ~/$ROOT/data/
  gunzip RXBC_20180122_INSERT_EXTRACT.dat.gz
  cd ~/$ROOT/
fi

# Copy the input file to the HDFS system
hdfs dfs -mkdir -p /user/hadoop/mongo_input
hdfs dfs -put ~/$ROOT/data/RXBC_20180122_INSERT_EXTRACT.dat /user/hadoop/mongo_input

# Delete contents of old output folder and the output folder itself
hdfs dfs -rmr /user/hadoop/mongo_output

# Disable the SLF4J jar for the duration of the run to prevent excessive logging
# by the Mongo Driver
sudo mv /usr/hdp/2.6.2.3-1/hadoop/lib/slf4j-log4j12-1.7.10.jar /usr/hdp/2.6.2.3-1/hadoop/lib/slf4j-log4j12-1.7.10.jarx
2>/dev/null

# User parameters to MongoLoad.jar are  hdfs:/path/to/input/file   hdfs:/nonexistent/output/folder
hadoop jar ~/$ROOT/dist/MongoLoad.jar /user/hadoop/mongo_input/RXBC_20180122_INSERT_EXTRACT.dat
/user/hadoop/mongo_output

# Restore the SLF4J driver
sudo mv /usr/hdp/2.6.2.3-1/hadoop/lib/slf4j-log4j12-1.7.10.jarx /usr/hdp/2.6.2.3-1/hadoop/lib/slf4j-log4j12-1.7.10.jar
2>/dev/null
```

By this time, the MapReduce operation is complete and we are sitting at the command prompt again, viewing the summary results.

```
18/02/11 06:20:12 INFO mapreduce.Job:  map 97% reduce 0%
18/02/11 06:20:36 INFO mapreduce.Job:  map 98% reduce 0%
18/02/11 06:21:00 INFO mapreduce.Job:  map 99% reduce 0%
18/02/11 06:21:25 INFO mapreduce.Job:  map 100% reduce 0%
18/02/11 06:21:36 INFO mapreduce.Job: Job job_1518238224323_0002 completed successfully
18/02/11 06:21:36 INFO mapreduce.Job: Counters: 30
        File System Counters
                FILE: Number of bytes read=0
                FILE: Number of bytes written=472302
                FILE: Number of read operations=0
                FILE: Number of large read operations=0
                FILE: Number of write operations=0
                WASB: Number of bytes read=43064741
                WASB: Number of bytes written=0
                WASB: Number of read operations=0
                WASB: Number of large read operations=0
                WASB: Number of write operations=0
        Job Counters
                Launched map tasks=3
                Rack-local map tasks=3
                Total time spent by all maps in occupied slots (ms)=5510326
                Total time spent by all reduces in occupied slots (ms)=0
                Total time spent by all map tasks (ms)=5510326
                Total vcore-milliseconds taken by all map tasks=5510326
                Total megabyte-milliseconds taken by all map tasks=8463860736
        Map-Reduce Framework
                Map input records=90669
                Map output records=0
                Input split bytes=591
                Spilled Records=0
                Failed Shuffles=0
                Merged Map outputs=0
                GC time elapsed (ms)=921
                CPU time spent (ms)=94260
                Physical memory (bytes) snapshot=854478848
                Virtual memory (bytes) snapshot=8520286208
                Total committed heap usage (bytes)=3193962496
        File Input Format Counters
                Bytes Read=43063559
        File Output Format Counters
                Bytes Written=0
sshuser@hn0-lummyh:~/DeepAzure_Project$
```

Summary

The data load did complete successfully and in parallel, but the performance was lackluster. Gains could be obtained by further optimizing as follows:

Rearrange data so that the MongoMap can access multiple data documents in one iteration – then a bulk insert operation could be leveraged which is much faster than single inserts

Break up the data file into smaller chunks that can be sent to more and more threads of execution for greater parallelism. The MongoLoader.java has provisions to control the splitting of the file, near the beginning.

Build out the MongoConnection object to be a singleton/static class with Connection Pooling so that repeated connections to the DB need not be made.

Increase the throughput of the CosmoDB/MongoDB from the default 1000 RU/s to something higher.

Pros

Code was easily implemented in Java, with minimal learning curve.

Data that was pulled straight from the DS Grid loads with minimal tweaking.

It was fun!

Cons

Requires a Java developer (such as myself) to implement the code; code must be compiled such that it runs in the Hadoop infrastructure.

Data Services Grid Infrastructure is built upon Perl code, but can easily run any stand-alone programs written in Perl, Java, C, C#, Shell Script, Python, Ruby, etc. As long as the work (or data) can be split up and run independently of the other, the grid infrastructure can be utilized for any program. Not readily apparent in Hadoop.

DS is staffed by many Perl developers who will need to relearn the Hadoop technology, and porting over the code from Data Services to the MS Azure Cloud would require a substantial rewrite.

Hadoop splits files based on file sizes rather than work unit or row count as the grid infrastructure does. It also feeds data one line at a time to the Map program whereas we can control exactly what gets fed in what chunks on the grid. There seems to be more restrictions on how data is moved through the process.

Links, References

2 Minute YouTube Introduction Video:

https://www.youtube.com/watch?v=4XZVUQicuIM

15 Minute YouTube Demonstration Video:

https://www.youtube.com/watch?v=YHRTkNVww3k

GitHub Repository Location:

https://github.com/lumkichi/DeepAzure_Project

Links for sources to 3rd party software and library files:

BitVise SSH

https://www.bitvise.com/ssh-client-download

RoboMongo 3T

https://robomongo.org/download

mongo-java-driver-3.6.1.jar (the java driver to connect to MongoDB)

https://oss.sonatype.org/content/repositories/releases/org/mongodb/mongodb-driver/3.6.1/

slf4j-nop-1.7.25.jar (used for suppressing mongo driver log output)

http://central.maven.org/maven2/org/slf4j/slf4j-nop/1.7.25/