Lab Manual

Course

BE Computer Engineering

Semester – VIII

Subject

Big Data Analytics

Code: 2180710

Prepared by,

Prof. Yogesh M. Kapuriya

Assistant Professor,

Computer Engineering

C. K. Pithawalla College of Engineering and Technology,

Surat

Index

Assignment 1

1.1 Understand and demonstrate List, Set and Map in Java.

1.2 Student basic information management using various collection types.

```
/*
 *
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */
package setlistmapdemo;

import java.util.*;
import java.io.InputStreamReader;
import java.io.BufferedReader;
import java.io.IOException;

/**
 *
 * @author Yogesh Kapuriya
 */
public class SetListMapDemo {

    /**
     * @param args the command line arguments
     */
    ArrayList students;
    BufferedReader ip;

    public SetListMapDemo() {
        this.students = new ArrayList();
        this.ip = new BufferedReader(new InputStreamReader(System.in));
    }

    public static void main(String[] args) throws IOException {
        // TODO code application logic here
        String enrollNo, stuName, contactNo, dob;
```

```
        SetListMapDemo MainProg = new SetListMapDemo();

        main_loop:
        while (true) {
            System.out.println("Enter your Choice");
            System.out.println("1. Add Student");
            System.out.println("2. Find Student");
            System.out.println("3. Update Student Detail");
            System.out.println("4. Delete Student");
            System.out.println("5. Get Count of Students");
            System.out.println("6. List all Students");
            System.out.println("7. Exit");
            String choice = MainProg.ip.readLine();
            int ch = Integer.parseInt(choice);
            switch (ch) {
                case 1:
                    System.out.println("Enter Enrollment No :");
                    enrollNo = MainProg.ip.readLine();
                    System.out.println("Enter Name :");
                    stuName = MainProg.ip.readLine();
                    System.out.println("Enter Contact No :");
                    contactNo = MainProg.ip.readLine();
                    System.out.println("Enter DOB :");
                    dob = MainProg.ip.readLine();
                    MainProg.addStudent(enrollNo, stuName, contactNo, dob);
                    break;
                case 2:
                    System.out.println("Enter Enrollment No of Student to be
Found :");
                    enrollNo = MainProg.ip.readLine();
                    HashMap s = MainProg.findStudent(enrollNo);
                    if (s.isEmpty()) {
                        System.out.println("Student not Found");
                    } else {
                        System.out.println("Student Detail : Enrollment No - "
+ s.get("EnrollNo") + ", Name : " + s.get("Name") + ", Contact No : " +
s.get("ContactNo") + ", DOB : " + s.get("DOB"));
                    }
                    break;
```

```
                case 3:
                    System.out.println("Enter Enrollment No of Student to be
Updated :");
                    enrollNo = MainProg.ip.readLine();
                    MainProg.updateStudent(enrollNo);
                    break;
                case 4:
                    System.out.println("Enter Enrollment No of Student to be
Deleted :");
                    enrollNo = MainProg.ip.readLine();
                    MainProg.deleteStudent(enrollNo);
                    break;
                case 5:
                    int count = MainProg.getStudentCount();
                    System.out.println("Total Students : " + count);
                    break;
                case 6:
                    System.out.println("List of Stundets");
                    MainProg.listStudents();
                    break;
                case 7:
                    break main_loop;
            }
        }
    }

    /**
     * Adds new Student to List of students
     *
     * @param enroll - Enrollment no of student
     * @param name - Name of student
     * @param contact - Contact no of student
     * @param dob - Date of birth in dd-mm-yyyy format
     */
    public void addStudent(String enroll, String name, String contact, String
dob) {
        HashMap student = new HashMap();
        student.put("EnrollNo", enroll);
        student.put("Name", name);
```

```
        student.put("ContactNo", contact);
        student.put("DOB", dob);
        this.students.add(student);
    }

    /**
     * Finds student with given enrollment no
     *
     * @param enroll
     * @return Student Object or empty object
     */
    public HashMap findStudent(String enroll) {
        HashMap stu;
        Iterator itr = students.iterator();
        while (itr.hasNext()) {
            stu = (HashMap) itr.next();
            if (enroll.equalsIgnoreCase((String) stu.get("EnrollNo"))) {
                System.out.println("Here");
                return stu;
            }
        }
        stu = new HashMap();
        return stu;
    }

    /**
     * Updates student detail
     *
     * @param enroll
     * @throws java.io.IOException
     */
    public void updateStudent(String enroll) throws IOException {
        HashMap stu = this.findStudent(enroll);
        if (stu.isEmpty()) {
            System.out.println("Student not Found");
        } else {
            String stuName, contactNo, dob;
            System.out.println("Student Current Detail : ");
```

```
            System.out.println("Name :" + stu.get("Name") + "New Name : ");
            stuName = this.ip.readLine();
            System.out.println("Contact No : " + stu.get("ContactNo") + "New
Contact No :");
            contactNo = this.ip.readLine();
            System.out.println("DOB :" + stu.get("DOB"));
            dob = this.ip.readLine();
            stu.replace("Name", stuName);
            stu.replace("ContactNo", contactNo);
            stu.replace("DOB", dob);
        }
        return;
    }

    /**
     * Delete student with given enrollment no
     * @param enroll
     */
    public void deleteStudent(String enroll) throws IOException {
        HashMap stu = this.findStudent(enroll);
        if (stu.isEmpty()) {
            System.out.println("Student not Found");
        } else {
            System.out.println("Student Detail : Enrollment No - " +
stu.get("Name") + ", Name : " + stu.get("ContactNo"));
            System.out.println("Are you sure? Type - Yes or No");
            String choice = this.ip.readLine();
            if (choice.equalsIgnoreCase("no")) {
                System.out.println("Operation cancelled !");
            } else {
                this.students.remove(this.students.indexOf(stu));
            }
        }
    }

    /**
     * Get count of students in student list
     *
     * @return
```

```
     */
    public int getStudentCount() {
        return this.students.size();
    }

    /**
     * List all students present in List
     */
    public void listStudents() {
        Iterator itr = students.iterator();
        while (itr.hasNext()) {
            HashMap s = (HashMap) itr.next();
            System.out.println("Student Detail : Enrollment No - " +
s.get("EnrollNo") + ", Name : " + s.get("Name") + ", Contact No : " +
s.get("ContactNo") + ", DOB : " + s.get("DOB"));
        }
    }
}
```

Assignment 2

2.1 MongoDB with PHP on Apache Web server on Ubuntu platform.

Step 1 — Adding the MongoDB Repository

MongoDB is already included in Ubuntu package repositories, but the official MongoDB repository provides most up-to-date version and is the recommended way of installing the software. In this step, we will add this official repository to our server.

Ubuntu ensures the authenticity of software packages by verifying that they are signed with GPG keys, so we first have to import they key for the official MongoDB repository.

```
sudo apt-key adv --keyserver hkp://keyserver.ubuntu.com:80 --recv EA312927
```

Next, we have to add the MongoDB repository details so apt will know where to download the packages from.

Issue the following command to create a list file for MongoDB.

```
echo "deb http://repo.mongodb.org/apt/ubuntu xenial/mongodb-org/3.2 multiverse" |
sudo tee /etc/apt/sources.list.d/mongodb-org-3.2.list
```

After adding the repository details, we need to update the packages list.

```
sudo apt-get update
```

Step 2 — Installing and Verifying MongoDB

Now we can install the MongoDB package itself.

```
sudo apt-get install -y mongodb-org
```

This command will install several packages containing latest stable version of MongoDB along with helpful management tools for the MongoDB server.

In order to properly launch MongoDB as a service on Ubuntu 16.04, we additionally need to create a unit file describing the service. A unit file tells systemd how to manage a resource. The most common unit type is a service, which determines how to start or stop the service, when should it be automatically started at boot, and whether it is dependent on other software to run.

We'll create a unit file to manage the MongoDB service. Create a configuration file named mongodb.service in the /etc/systemd/system directory using nano or your favorite text editor.

sudo nano /etc/systemd/system/mongodb.service

Paste in the following contents, then save and close the file.

```
[Unit]
Description=High-performance, schema-free document-oriented database
After=network.target
[Service]
User=mongodb
ExecStart=/usr/bin/mongod --quiet --config /etc/mongod.conf
[Install]
WantedBy=multi-user.target
```

This file has a simple structure:

- The Unit section contains the overview (e.g. a human-readable description for MongoDB service) as well as dependencies that must be satisfied before the service is started. In our case, MongoDB depends on networking already being available, hence network.target here.
- The Service section how the service should be started. The User directive specifies that the server will be run under the mongodb user, and the ExecStart directive defines the startup command for MongoDB server.
- The last section, Install, tells systemd when the service should be automatically started. The multi-user.target is a standard system startup sequence, which means the server will be automatically started during boot.

Next, start the newly created service with systemctl.

sudo systemctl start mongodb

While there is no output to this command, you can also use systemctl to check that the service has started properly.

sudo systemctl status mongodb

The last step is to enable automatically starting MongoDB when the system starts.

sudo systemctl enable mongodb

The MongoDB server now configured and running, and you can manage the MongoDB service using the systemctl command (e.g. sudo systemctl mongodb stop, sudo systemctl mongodb start).

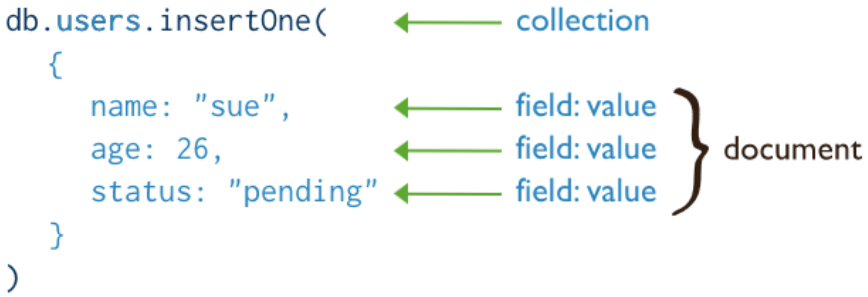
2.2 Basic CRUD operations and aggregate functions in MongoDB.

Create Operations

Create or insert operations add new documents to a collection. If the collection does not currently exist, insert operations will create the collection. MongoDB provides the following methods to insert documents into a collection:

- `db.collection.insertOne()` New in version 3.2
- `db.collection.insertMany()` New in version 3.2

In MongoDB, insert operations target a single collection. All write operations in MongoDB are atomic on the level of a single document.



Read Operations

Read operations retrieves documents from a collection; i.e. queries a collection for documents. MongoDB provides the following methods to read documents from a collection:

- `db.collection.find()`

You can specify query filters or criteria that identify the documents to return.

Update Operations

Update operations modify existing documents in a collection. MongoDB provides the following methods to update documents of a collection:

- `db.collection.updateOne()` New in version 3.2
- `db.collection.updateMany()` New in version 3.2
- `db.collection.replaceOne()` New in version 3.2

In MongoDB, update operations target a single collection. All write operations in MongoDB are atomic on the level of a single document.

You can specify criteria, or filters, that identify the documents to update. These filters use the same syntax as read operations.

Delete Operations

Delete operations remove documents from a collection. MongoDB provides the following methods to delete documents of a collection:

- `db.collection.deleteOne()` New in version 3.2
- `db.collection.deleteMany()` New in version 3.2

In MongoDB, delete operations target a single collection. All write operations in MongoDB are atomic on the level of a single document.

You can specify criteria, or filters, that identify the documents to remove. These filters use the same syntax as read operations.

2.3 Web based application for Student registration using PHP and MongoDB.

Fields to be manage for student are : <enrolment_no, name, email, contact_no, date_of_birth, branch, address, tution_fees> Implement the following,

- Register new student.
- Get list of registered students.
- Update registration details.
- Generate branch wise student registration summary.
- Delete registration for any student.

Assignment 3

3.1 Installation of Hadoop on Ubuntu platform

INSTALL SSH

You may need to update the repository before openssh is detected

```
sudo apt-get update

sudo apt-get install openssh-server
```

Give yes if asked for dependencies

Generate keys

```
ssh-keygen                [PRESS ENTER]

ENTER FILE NAME:          [EMPTY]

ENTER PASSPHRASE:         [EMPTY]

ssh localhost             [GIVE YES]

password:hadoop
```

It gets connected

```
cat ~/.ssh/id_rsa.pub >> ~/.ssh/authorized_keys

ssh localhost
```

Connected without password!!!!!!

INSTALL JAVA

For offline install: place tar.gz file in downloads folder

```
sudo mkdir -p /usr/lib/jvm/

sudo tar xvf ~/Downloads/jdk-7u67-linux-x64.tar.gz -C /usr/lib/jvm

(for multinode, copy same folder in all machines using command: sudo rcp -r
<username>@<machine_name>:/usr/lib/jvm/ /usr/lib/)
```

```
cd /usr/lib/jvm

sudo ln -s jdk1.7.0_67 java-1.7.0-sun-amd64

sudo update-alternatives --config java
```

Now we need to set path

```
sudo nano $HOME/.bashrc

export JAVA_HOME="/usr/lib/jvm/jdk1.7.0_67"

export PATH="$PATH:$JAVA_HOME/bin"

[ctrl+O to save, Press ENTER to confirm, ctrl+x to exit]
```

To test

```
exec bash

$PATH
```

The new path u have added should also appear here

Change directory to home

```
cd ~
```

INSTALLING HADOOP

Place hadoop in home directory

(can't do it in downloads as we dont have permission to create folders/files in that directory)

```
cd ~

sudo mkdir -p /usr/local/hadoop/

sudo tar xvf ~/hadoop-1.2.1-bin.tar.gz -C /usr/local/hadoop

(for multinode, copy same folder in all machines using command: sudo rcp -r
<username>@<machine_name>:/usr/local/hadoop/ /usr/local/)

password: hadoop
```

Check ---

```
cd /usr/local/hadoop

ls
```

Change back to home directory

```
cd ~
```

Setting the path

```
sudo nano $HOME/.bashrc

[GOTO LAST LINE AND PASTE]

export HADOOP_PREFIX=/usr/local/hadoop

export PATH=$PATH:$HADOOP_PREFIX/bin

[ctrl+O to save , Press ENTER to confirm, ctrl+x to exit]
```

To test

```
exec bash

$PATH
```

The new path u have added should also appear here

CONFIGURING HADOOP ENVIRONMENT

```
cd /usr/local/hadoop/conf

sudo nano hadoop-env.sh
```

Add

```
export JAVA_HOME=/usr/lib/jvm/java-1.7.0-sun-amd64
```

Search

```
#export HADOOP_OPTS

export HADOOP_OPTS=-Djava.net.preferIPv4Stack=true

[Ctrl+O to save, ENTER, Ctrl+X to exit]
```

CONFIGURATION

Use namenode name instead of hnname

Especially for multinode deployment provide the name of namenode and not your system name

sudo nano core-site.xml

```
<configuration>
  <property>
      <name>fs.default.name</name>
      <value>hdfs://coed161:10001</value>
  </property>
  <property>
      <name>hadoop.tmp.dir</name>
      <value>/usr/local/hadoop/tmp</value>
  </property>
</configuration>
```

[ctrl+o, ENTER, ctrl+x]

sudo nano mapred-site.xml

```
<configuration>
  <property>
      <name>mapred.job.tracker</name>
      <value>coed161:10002</value>
  </property>
</configuration>
```

Create temp directory

sudo mkdir /usr/local/hadoop/tmp

pwd:hadoop

sudo chown <username> /usr/local/hadoop/tmp

sudo chown <username> /usr/local/hadoop

Formatting the DFS

Don't do if you are creating a multi-node deployment

```
hadoop namenode -format
```

Check for success message

Start all process

```
start-all.sh
```

Java process status: shows all running process

```
jps
```

To see the details from gui web interface

Go to firefox

Hadoop administration

Change the namenode name accordingly

```
http://coed161:50070/dfshealth.jsp
```

Look for live nodes and browse the dfs file system

Hadoop job tracker

Change the namenode name accordingly

```
http://coed161:50030/jobtracker.jsp
```

To stop all process

```
stop-all.sh
```

Firewall install

```
sudo iptables -L -n

sudo iptables -I INPUT -p tcp --dport 10001  -j ACCEPT

sudo iptables -I INPUT -p tcp --dport 10000 -j ACCEPT

sudo iptables -I INPUT -p tcp --dport 10002 -j ACCEPT

sudo iptables -I INPUT -p tcp --dport 50010 -j ACCEPT

sudo iptables -L -n
```

CD into the ip dir and run the following command

```
sudo dpkg -i *.deb
```

Asks for saving current ipv4 rules give yes

Asks for saving current ipv6 rules give yes

Benchmarking demo

copy the hadoop-examples-1.2.1.jar file to the home directory in a folder called binary: where we will store all jar files and executables

```
hadoop jar /home/anand/binary/hadoop-examples-1.2.1.jar teragen 10000000
/data/input
```

Terasort demo (should run teragen to generate input data first)

Change output directory each time or remove the folder after each run else it will fail

```
hadoop jar /home/anand/binary/hadoop-examples-1.2.1.jar terasort  /data/input
/data/output
```

To remove directories

Change the namenode name accordingly

```
hadoop fs -rmr hdfs://decoy:10001/data/input

hadoop fs -rmr hdfs://decoy:10001/data/output
```

Hadoop Word Count Demo

to copy the input data to hadoop hdfs

```
hadoop fs -put /home/anand/datanew/1/ hdfs://decoy:10001/data/input/book/

hadoop fs -put /home/anand/datanew/2/ hdfs://decoy:10001/data/input/purchases/

hadoop fs -put /home/anand/datanew/3/ hdfs://decoy:10001/data/input/weather/

hadoop fs -put /home/anand/datanew/4/ hdfs://decoy:10001/data/input/accesslog/
```

```
hadoop jar /home/anand/binary/hadoop-examples-1.2.1.jar wordcount
hdfs://decoy:10001/data/input/book/ hdfs://decoy:10001/data/output/book/outputmine
```

3.2 Understand the overall programming architecture using Map Reduce API

Hadoop MapReduce can be defined as a software programming framework used to process big volume of data (in terabyte level) in a parallel environment of clustered nodes. The cluster consists of thousands of nodes of commodity hardware. The processing is distributed, reliable and fault tolerant. A typical MapReduce job is performed according to the following steps:

1. Split the data into independent chunks based on key-value pair. This is done by Map task in a parallel manner.
2. The output of the Map job is sorted based on the key values
3. The sorted output is the input to the Reduce job. And then it produces the final output to the processing and returns the result to the client.

MapReduce Framework

The Apache Hadoop MapReduce framework is written in Java. The framework consists of master-slave configuration. The master is known as JobTracker and the slaves are known as TaskTrackers. The master controls the task processed on the slaves (which are nothing but the nodes in a cluster). The computation is done on the slaves. So the compute and storages nodes are the same in a clustered environment. The concept is ' move the computation to the nodes where the data is stored', and it makes the processing faster.

MapReduce Processing

The MapReduce framework model is very lightweight. So the cost of hardware is low compared with other frameworks. But at the same time, we should understand that the model works efficiently only in a distributed environment as the processing is done on nodes where the data resides. The other features like scalability, reliability and fault tolerance also works well on distributed environment.

MapReduce Implementation

The following are the different components of the entire end-to-end implementation.

- The client program that is the driver class and initiates the process
- The Map function that performs the split using the key-value pair.
- The Reduce function that aggregate the processed data and send the output back to the client.

3.3 Develop Map Reduce Work Application

WordCount in JAVA.: https://hadoop.apache.org/docs/r1.2.1/mapred_tutorial.html

Assignment 4

4.1 Installation of Hive and Pig on Hadoop Ecosystem.

Installation of Hive

https://cwiki.apache.org/confluence/display/Hive/AdminManual+Installation

Installation and Getting Started with Pig

http://pig.apache.org/docs/r0.16.0/start.html#req

4.2 Creating the HDFS tables and loading them in Hive and learn joining of tables in Hive

There are 2 types of tables in Hive, Internal and External. Case study described on following blog describes creation of table, loading data in it, creating views, indexes and dropping table on weather data.

https://www.dezyre.com/hadoop-tutorial/apache-hive-tutorial-tables

Assignment 5

5.1 Installation of Apache Spark.

Download latest version

http://spark.apache.org/docs/latest/

Installation in standalone mode

http://spark.apache.org/docs/latest/spark-standalone.html

5.2 Implement and demonstrate WordCount application in Apache Spark.

Python Script

```
counts = text_file.flatMap(lambda line: line.split(" ")) \
             .map(lambda word: (word, 1)) \
             .reduceByKey(lambda a, b: a + b)
counts.saveAsTextFile("hdfs://...")
```

Java Program

```
JavaRDD<String> textFile = sc.textFile("hdfs://...");
JavaPairRDD<String, Integer> counts = textFile
    .flatMap(s -> Arrays.asList(s.split(" ")).iterator())
    .mapToPair(word -> new Tuple2<>(word, 1))
    .reduceByKey((a, b) -> a + b);
counts.saveAsTextFile("hdfs://...");
```