Implementing Billing Services on a Hadoop Distributed System

Elaine Thai, Mowhebat Bazargani, Julius Aguma
University of California, Irvine
{thaiem, bazargam, jaguma}@uci.edu

***Abstract*—We implemented a billing service on distributed systems where students are the clients that want to send their payments to their university which hosts a server comprised of a local and cloud hadoop node. The payment request sent by the student includes their full name and the payment amount. Through the use of sockets, the client sends these payment requests to the server. The server then performs verification of student information and payment before updating the student's fee record (stored in a CSV file) accordingly and sends a confirmation upon success.**

***Index Terms*—hadoop, client-server architecture, distributed system**

I. Introduction

A. Project Motivation

From the distributed systems class, we learned how complex it is to implement reliable group communication in an environment of diverse devices running different operating systems. It is rather common to have school systems fail in and around fee deadlines when students rush to pay their fees at the last minutes creating a server overload. Additionally, accuracy and consistency are of huge priority in a billing service as the results could be disruptive to the students' routines and ability to attend class.

B. Project Objective

We, therefore set out to implement a multi-node client-server architecture that provides billing services to students as the clients and the university as the server. We hoped to have a multi-node server spread out over different systems with a master and slaves nodes as a way to replicate university systems where you have a master say the university cashier's office and slaves being the different school departments that coordinate with the cashier office to clear student fees.

II. Architecture

We have one Hadoop cluster with one master node and two slave nodes. One local machine act as the master while another and a cloud node are the slaves. The client side of the system runs on a local machine while the server side operates on the Hadoop cluster. The database (in the form of a CSV file) is stored in the master node of the Hadoop cluster, including students' name and the amount of money they owe. When requests are received from the client (which is another local machine), the Hadoop cluster splits the works between the nodes. The split is done automatically in the hadoop configuration with the master node being the namenode and

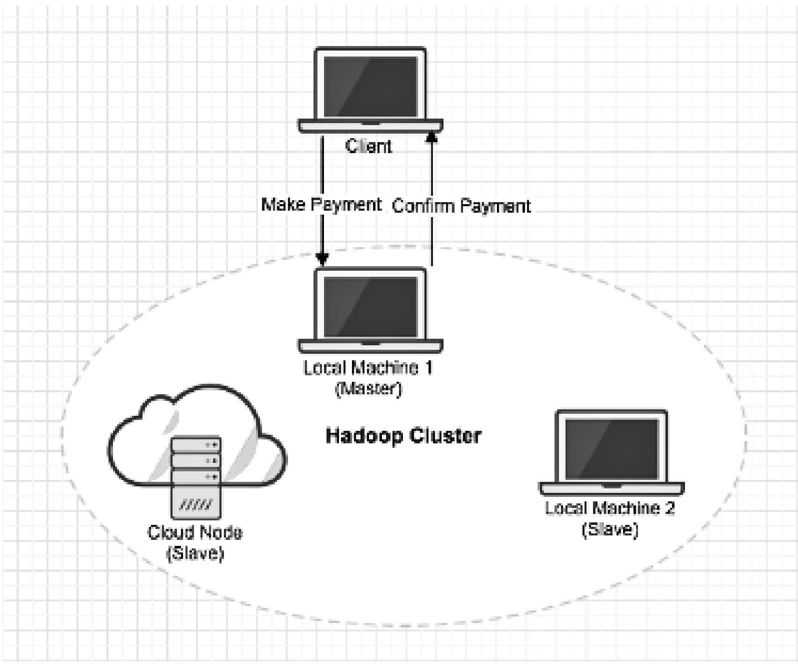
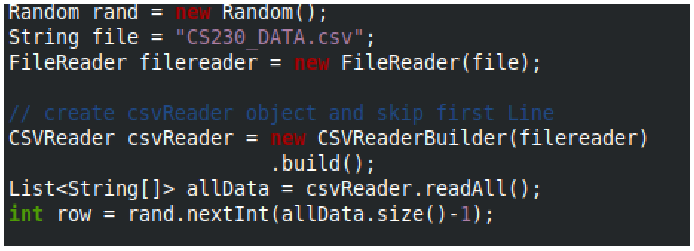


Fig. 1. Client to Multi-node Server Architecture

resource manager while the slaves are the datanodes. In the limitations sections, we will discuss the obstacles that hindered our ability to full utilize and expand this architecture.

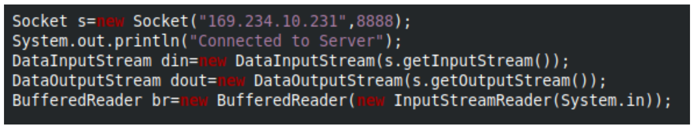
A. Client

The client is a simple java script that picks a random test student and send the student's information to the server over a TCP socket with a hard coded port. Below, we are code snippets from this script.



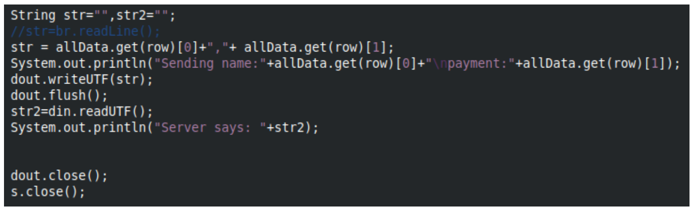
```
Random rand = new Random();
String file = "CS230_DATA.csv";
FileReader filereader = new FileReader(file);

// create csvReader object and skip first Line
CSVReader csvReader = new CSVReaderBuilder(filereader)
                        .build();
List<String[]> allData = csvReader.readAll();
int row = rand.nextInt(allData.size()-1);
```

Fig. 2. Client code snippet: Randomly picked student



```
Socket s=new Socket("169.234.10.231",8888);
System.out.println("Connected to Server");
DataInputStream din=new DataInputStream(s.getInputStream());
DataOutputStream dout=new DataOutputStream(s.getOutputStream());
BufferedReader br=new BufferedReader(new InputStreamReader(System.in));
```

Fig. 3. Client code snippet: Client connects to server socket



```
String str="",str2="";
//str=br.readLine();
str = allData.get(row)[0]+","+ allData.get(row)[1];
System.out.println("Sending name:"+allData.get(row)[0]+"\npayment:"+allData.get(row)[1]);
dout.writeUTF(str);
dout.flush();
str2=din.readUTF();
System.out.println("Server says: "+str2);


dout.close();
s.close();
```

Fig. 4. Client code snippet: Client sends and receives messages from server

B. Server

The server is implemented as a hadoop multi-node cluster [1] with one master node and two slaves. The nodes employ ssh as a communication channel and run on top of the java virtual machine. The master splits the workload by namenodes and datanodes. In our set up, the master is the namenode and the slaves are datanodes. The master is also a resource manager and the slaves are nodemanagers. Below we give more detail on the individual nodes.

1) Master node: The master node runs on a local machine with configuration of how to split the work set in a worker file. The master creates an ssh rsa key that is then distributed to the slaves to facilitate passwordless communication. After keys are shared, and all nodes configured, we simply run 'start-all.sh' to start the namenode, datanodes, resource manager, and nodemanagers. If all is successful, the process should show up in the the terminal with command 'jps'.

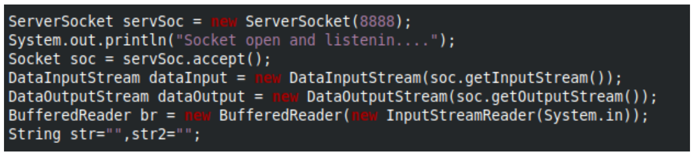
2) local slave node: The local slave node runs on a local machine and is remotely started by the server. Alot of the configuration for this node is similar to the master with the exception of the hostname that identify this machine as a slave.

3) cloud slave node: For our cloud node, we set up a single node EC2 node on the Amazon AWS servers. WE used the wizard to create a default instance with ssh capabilities, Using a key pair, we log into the cloud from a local machine through ssh and then configure the instance as a hadoop slave. Getting the configuration right was a crucial time consuming task.

4) Server java program: The server program is also a simple program that initiates a socket listening for client requests on port 8888. When a client sends over student info, the server uses a third party library(openscv) to access the csv file to find the student record, validate the information and update the payment value. Here are some code snippets.

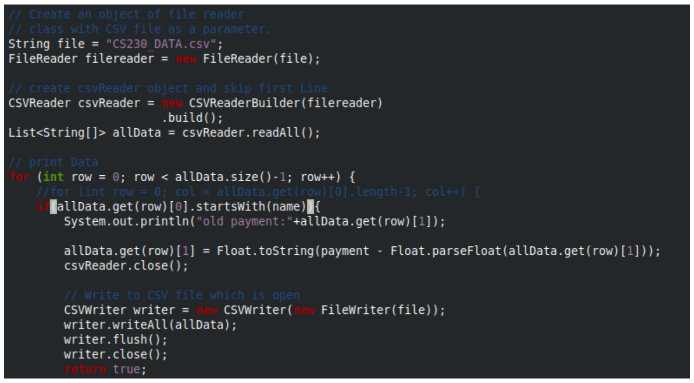
III. Project Evaluation

Much of our work was in the configuration of the multi-cluster hadoop system. We encountered many obstacles that will be discussed in the next section. Below we run the hadoop multi node cluster then communicate between the client and



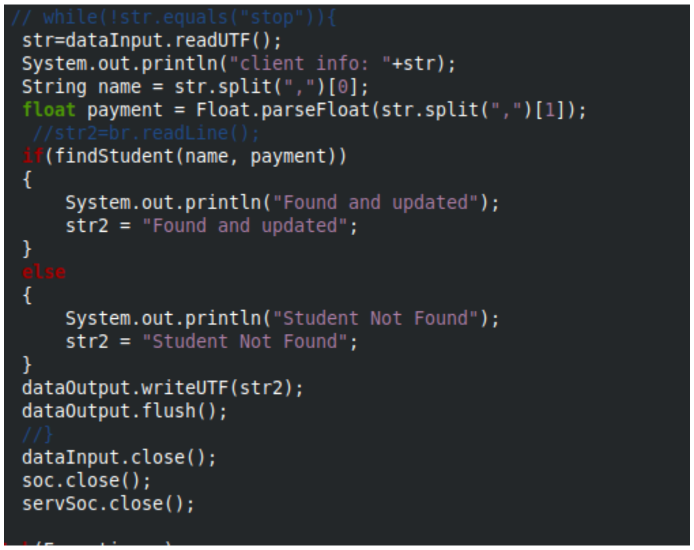
```
ServerSocket servSoc = new ServerSocket(8888);
System.out.println("Socket open and listenin....");
Socket soc = servSoc.accept();
DataInputStream dataInput = new DataInputStream(soc.getInputStream());
DataOutputStream dataOutput = new DataOutputStream(soc.getOutputStream());
BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
String str="",str2="";
```

Fig. 5. Server code snippet: Server socket creation



```
// Create an object of file reader
// class with CSV file as a parameter.
String file = "CS230_DATA.csv";
FileReader filereader = new FileReader(file);

// create csvReader object and skip first Line
CSVReader csvReader = new CSVReaderBuilder(filereader)
                      .build();
List<String[]> allData = csvReader.readAll();

// print Data
for (int row = 0; row < allData.size()-1; row++) {
    //for (int row = 0; col < allData.get(row)[0].length-1; col++) {
    if(allData.get(row)[0].startsWith(name)){
        System.out.println("old payment:"+allData.get(row)[1]);

        allData.get(row)[1] = Float.toString(payment - Float.parseFloat(allData.get(row)[1]));
        csvReader.close();

        // Write to CSV file which is open
        CSVWriter writer = new CSVWriter(new FileWriter(file));
        writer.writeAll(allData);
        writer.flush();
        writer.close();
        return true;
```

Fig. 6. Server code snippet: Server accessing csv file

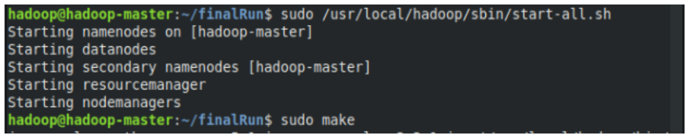


```
// while(!str.equals("stop")){
 str=dataInput.readUTF();
 System.out.println("client info: "+str);
 String name = str.split(",")[0];
 float payment = Float.parseFloat(str.split(",")[1]);
  //str2=br.readLine();
 if(findStudent(name, payment))
 {
     System.out.println("Found and updated");
     str2 = "Found and updated";
 }
 else
 {
     System.out.println("Student Not Found");
     str2 = "Student Not Found";
 }
 dataOutput.writeUTF(str2);
 dataOutput.flush();
 //}
 dataInput.close();
 soc.close();
 servSoc.close();
```

Fig. 7. Server code snippet: Server socket receiving and sending messages to client

server nodes. We planned to test the system's reliability by shutting down on node and seeing how the system reacts but due to time constraints, we did not meet that goal.

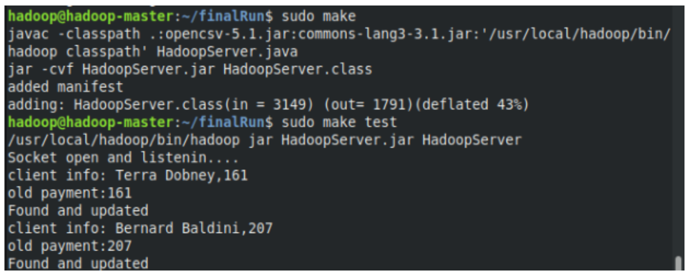
We begin by starting the namenode, datanode, resource manager, and nodemanagers using the hadoop command, 'start-all.sh'.



```
hadoop@hadoop-master:~/finalRun$ sudo /usr/local/hadoop/sbin/start-all.sh
Starting namenodes on [hadoop-master]
Starting datanodes
Starting secondary namenodes [hadoop-master]
Starting resourcemanager
Starting nodemanagers
hadoop@hadoop-master:~/finalRun$ sudo make
```

Fig. 8. Starting multi-node Hadoop system

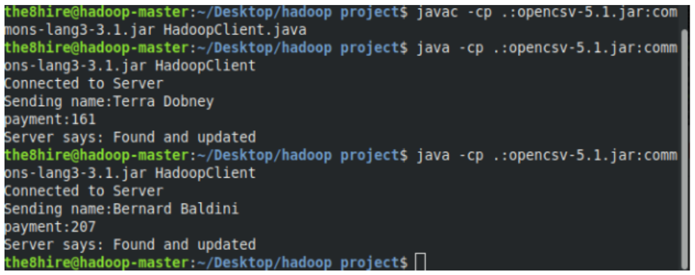
Then using a makefile, we compile and run the server java script on the hadoop system with the ’hadoop program’ script.



```
hadoop@hadoop-master:~/finalRun$ sudo make
javac -classpath .:opencsv-5.1.jar:commons-lang3-3.1.jar:'/usr/local/hadoop/bin/
hadoop classpath' HadoopServer.java
jar -cvf HadoopServer.jar HadoopServer.class
added manifest
adding: HadoopServer.class(in = 3149) (out= 1791)(deflated 43%)
hadoop@hadoop-master:~/finalRun$ sudo make test
/usr/local/hadoop/bin/hadoop jar HadoopServer.jar HadoopServer
Socket open and listenin....
client info: Terra Dobney,161
old payment:161
Found and updated
client info: Bernard Baldini,207
old payment:207
Found and updated
```

Fig. 9. Server run on hadoop

The Socket is created and wait for client resquests to which it responds.



```
the8hire@hadoop-master:~/Desktop/hadoop project$ javac -cp .:opencsv-5.1.jar:com
mons-lang3-3.1.jar HadoopClient.java
the8hire@hadoop-master:~/Desktop/hadoop project$ java -cp .:opencsv-5.1.jar:comm
ons-lang3-3.1.jar HadoopClient
Connected to Server
Sending name:Terra Dobney
payment:161
Server says: Found and updated
the8hire@hadoop-master:~/Desktop/hadoop project$ java -cp .:opencsv-5.1.jar:comm
ons-lang3-3.1.jar HadoopClient
Connected to Server
Sending name:Bernard Baldini
payment:207
Server says: Found and updated
the8hire@hadoop-master:~/Desktop/hadoop project$
```

Fig. 10. Client requests

A. *Results and Limitations*

B. *Results*

We were able to successfully set up a hadoop multi-cluster with a master and two slaves. We then successfully ran client and server scripts to facilitate client requests for payment that were successfully handled by the server. While it is not shown here, we believe that given our configuration, this system can still operate even if one slave fails. We believe we achieved reliability and with more time we could have demonstrated that too.

C. *Limitation*

Unfortunately, we faced many implementation issues with the use of Hadoop and connecting local clusters on separate machines. When trying to establish other machines as slave nodes in the Hadoop cluster, there were constant issues with trying to ssh copy id for the key files and connection errors as well. We attempted to solve these errors by adjusting permissions, firewall settings, and more, but these problems persisted. There was an instance where we managed to establish local machine 2’s Hadoop node as the slave node for local machine 1’s Hadoop cluster, but following that, we still ran into the same error.

There were also implementation issues when trying to run the client code on local machine 3 and connecting to local machine 1 on the server side with the use of a socket and a specific port 8888. Originally, there was a connection timeout error. Firewall settings were modified, but reruns would still result in this error. We hypothesized that perhaps the problem was local machine 1’s location and its associated firewall and tried to see if things would change if in a different location with a different IP–if this would allow a connection. When we tried running the client code again, this time there was a connection refused error.

Each group member has a different OS (Windows, Linux, and Mac) which sometimes proved to be a temporary hindrance when trying to troubleshoot issues, although Linux and Mac were similar enough that there were not many problems in that regard.

IV. Conclusion

We implemented a billing service on distributed systems where students are the clients that want to send their payments to their university which hosts a server comprised of a local and cloud hadoop node. The payment request sent by the student includes their full name and the payment amount. Through the use of sockets, the client sends these payment requests to the server. The server then performs verification of student information and payment before updating the student’s fee record (stored in a CSV file) accordingly and sends a confirmation upon success.

Possible extensions to this project would involve scaling up. We would require a larger sample of random data (students and owed fees) than 1000. A more ideal size would be 10,000 rows of data or larger. In addition, we would like to be able to add more slave nodes to the server cluster to allow for a more balanced load of work processing and updating the data on each node. The client could also be turned into its own Hadoop cluster with some number of nodes to more efficiently send a larger amount of requests to the server.

V. Contributions

A. *Elaine Thai*

Elaine implemented early cloud cluster instances, wrote up the project proposal, worked on the project slides and wrote up the first draft of the project report. We intnded to run the client on Elaine’s local machine but firewall permissions prevented that.

B. *Mowhebat Bazargani*

Mowhebat implemented one of the local slave nodes, worked on the project slides and contributed to the project proposal and project report. We intended to run the hadoop slave 1 on Mowhebat’s local machine but firewalls and network filter blocks prevented that.

C. *Julius Ceasar Aguma*

Ceasar was responsible for setting up the final cloud instance, wrote the server code and final draft of the project report. We intended to use ceasar’s laptop as the hadoop master but ended up having to run both server and client on it due to obstacles with multi-machine connection using ip addresses.

References

[1] David Júlio. How to setup hadoop 3.2.1 multi-node cluster on ubuntu 18.04. https://medium.com/@davidsjulio97/how-to-setup-hadoop-3-2-1-multi-node-cluster-on-ubuntu-18-04-d0b3d82abc13, 2019.