**Team 2 - CS 441 Project Documentation**

Team number 2

Team members:

Pratik Anil Kshirsagar, Graduate, 671863272, pkshir2@uic.edu

Gurpreet Kaur Chabada, Graduate, 651102968, gchaba2@uic.edu

Ashwani Khemani, Graduate, 660570981, akhema2@uic.edu

Dipankar Ghosh, Graduate, 652811050, dghosh6@uic.edu

Riccardo Pressiani, Graduate, 650547946, rpress4@uic.edu

Giovanni Agugini Bassi, Graduate, 659013265, gagugi2@uic.edu

Mayuri Kumari, Graduate, 672159677, [mkumar29@uic.edu](mailto:mkumar29@uic.edu)

The contents of the document are organised in the following order:

1. Brief overview of our system and methodology
2. What we did in the project with commands and description
3. Summarised steps to test the application

***Brief overview of our system and methodology:***

We decided to run our Monte Carlo simulator on a Kubernetes cluster which consists of 1 Kubernetes Master and 6 Kubernetes slaves. We run a Spark cluster integrated with this Kubernetes cluster such that there is 1 Spark master and the rest are the Spark workers. Our entire cluster is made up of Raspberry Pi 3s. We made docker images for each role (master & slave) and we ran the containers on the K8s cluster. We use Amazon Simple Storage Service (S3) for input and output directories (globally accessible locations). The Monte Carlo Simulator application is fed as a spark (submit) job to the cluster to produce desired output.

The input to our simulator is a randomly selected list of stocks, our total fund amount in USD, and a time period of these recorded stocks. We obtained this stock data through Alpha Vantage (https://www.alphavantage.co/). They provide APIs to access historical stock data in various formats and time series. We allocate our fund using random number generator to purchase stocks at the beginning. The simulation records gains and losses as time progresses and sell or buy stocks (buys randomly) due to current situation of the “market”. As it has ended, we record the gains and losses and we run the simulation as many number of times as we want.

A brief summary of steps we followed for this project:

1. Get financial data.
2. Transfer the csv files to s3 bucket which will be the input folder while running this application on spark.
3. Compile the simulator program and generate the jar which will be run on the pis
4. Setup kubernetes cluster on the pis
5. Generate docker images for running spark on pis
6. Setup spark cluster on the pis by pulling the docker images
7. Run the jar generated in step 3 on the spark cluster.
8. The output files will be stored on a remote location, s3 bucket directory.
9. Output can also be accessed in the log file of the master.

The above steps will be described in detail along with the commands below:

**Get financial data:**

Run the shell file with the command: sh downloadHistoricalData.sh

It calls the python script in the file downloadData.py which reads the company names present in companies\_list.txt file and makes a csv file for each company.

These csv files are stored in the stockData directory which should be the INPUT directory to the spark program. Transfer this stockData directory to a s3 bucket directory.

Description of the python file:

Historical stock market data are retrieved using a Python script which takes as input the name of the company and the time interval we are interested in. Data are retrieved as CSV files from Alpha Vintage (<https://www.alphavantage.co/>). The Python script provided parses and edits the downloaded files returning as output CSV files in the format required by the Monte Carlo simulator application. Moreover, a column is added to the CSV files in which the calculated change in price of the stock option is reported.

The output files contains the following columns: timestamp, open price, highest price, lowest price, closing price, volume, adjusted closing price, stock name, change in price.

The script provided is written with Python 2.7 and the *requests* library must be installed via *pip* before running the script.

**Generate jar:**

Generate the jar by running package command and skipping the tests for the monte carlo simulation program.

Move this jar to the directory from which we will run spark. This directory name is runningSpark for us.

**Monte Carlo simulation logic :**

1. Get financial data using the REST api.
2. We create a new csv file for each company specified in the company list file.
3. We compute change in price percentage per day for each company and store it in the csv file.
4. Formula for change in price percentage is ((close-open)/open)\*100
5. We run the simulation on these csv files.
6. We invest total amount X in 4 companies. The companies are chosen with uniform real distribution. X is distributed equally between these 4 companies.
7. The simulation begins on day 1 of the dates specified in the date range (see the python file).
8. For each date, total running amount for our investment is computed (for of all companies). If the computed value indicates that we made a loss today, we sell all the stocks and reinvest the money we obtain by selling them.
9. The reinvestment logic randomly selects 4 companies and equally distributes the current amount between them.
10. We run this simulation for the entire date range and display the losses or profits that could have been obtained.

***Project overview with commands and description***

**Setup the spark cluster and test our Monte Carlo Simulator application**

We will first need to the assemble the Raspberry Pi cluster in order to run the Monte Carlo Simulator.

Please follow the below steps in sequential order to make the Pi cluster ready for our simulation:

1. We will be using 7 Pi’s to setup the kubernetes cluster on it .
2. Setting up a Kubernetes cluster :

**\*Assembly of our physical Raspberry Pi Cluster:**

The number of devices = 7 Raspberry Pi 3's

Each RPi is flashed with HypriotOS, version 1.6, located at https://github.com/hypriot/image-builder-rpi/releases

To burn the OS onto the SD cards, we used Etcher, located at https://etcher.io/.

HypriotOS comes with Linux Kernel 4.4.50 for ARMv6 and ARMv7 and inbuilt:

Docker Engine 17.09.0-ce (officially build by Docker)

Docker Machine 0.12.2

Docker Compose 1.16.1

Every RPi was given a unique hostname by editing the init file at the SD card boot partition /boot/device-init.yaml

We used the Pis in headless mode, during the entire scope of the homework,

through ssh. We used the following commands to set up a Kubernetes cluster.

**#Performing the following on every Pi.**

* # Trust the Kubernetes APT key and add the official APT Kubernetes repository on every Pi.

sudo su

curl -s https://packages.cloud.google.com/apt/doc/apt-key.gpg | apt-key add -

echo "deb http://apt.kubernetes.io/ kubernetes-xenial main" > /etc/apt/sources.list.d/kubernetes.list

* # Install Kubernetes kubeadm on every Pi.

apt-get update && apt-get install -y kubeadm

**# Performing the following on the master Pi.**

* sudo su
* # Reset the Kubernetes cluster after fresh install.

kubeadm reset

* # Initialise Kubernetes on the master node.

kubeadm init --pod-network-cidr 10.244.0.0/16

* # Run the following as a regular user.

sudo cp /etc/kubernetes/admin.conf $HOME/

sudo chown $(id -u):$(id -g) $HOME/admin.conf

export KUBECONFIG=$HOME/admin.conf

* # Setup flannel v0.7.1 as the Pod network driver

curl -sSL https://rawgit.com/coreos/flannel/v0.7.1/Documentation/kube-flannel-rbac.yml | kubectl create -f -

curl -sSL https://rawgit.com/coreos/flannel/v0.7.1/Documentation/kube-flannel.yml | sed "s/amd64/arm/g" | kubectl create -f -

* # set accept in iptables for all requests

sudo iptables -P FORWARD ACCEPT

**# Perform following on every slave node to include it in the cluster.**

# The command will resemble something like this.

kubeadm join --token=bb14ca.e8bbbedf40c58788 192.168.0.34

sudo iptables -P FORWARD ACCEPT

* # On the master, perform the following.

# check status of nodes in the cluster.

kubectl get nodes

3. Once the kubernetes cluster is up and running and all the nodes are in the ready state, we can start working on running spark required to test our Monte Carlo Simulator .

**\*Creating the Kubernetes pods with Spark master and slaves.**

Inside the clone of this repo on the master pi(riccardo) run the following commands:

* cd runningSpark
* kubectl create -f spark-master.yaml

//This YAML will contain kind: Deployment, replicas: 1, image: <docker\_hub\_username>/<docker\_hub\_image\_name>, imagePullPolicy: "Always" -> to pull the latest image from Docker hub everytime, args: './start-master.sh; sleep infinity')

* kubectl create -f spark-master-service.yaml

//This YAML will contain kind: Service, spark port:

<Same\_as\_containerPort\_set\_in\_spark-master.yaml>

* kubectl create -f spark-worker.yaml

// This YAML will contain kind: Deployment, replicas: <atleast\_#RPIs-2>, image: <docker\_hub\_username>/<docker\_hub\_image\_name>,

imagePullPolicy: "IfNotPresent" (since latest image is already pulled using spark-master.yaml), containerPort: <containerPort\_set\_in\_spark-master.yaml\_+\_1>, args: './start-worker.sh; sleep infinity'

4. Once the spark cluster is up and running with the one of the nodes as the master and others as workers ready to run tasks assigned by the master , we can now test our application on it . Run the below command to run the monte carlo simulator application on the spark cluster .

* \*Enter the spark master pod shell.

kubectl get pods

kubectl exec -it <spark\_master\_pod\_name> bash

* \*Run Spark job inside Spark master pod.

spark-submit --properties-file s3.properties --class <class\_name> --master spark://spark-master:7077 --num-executors 1 --driver-memory 512m --executor-memory 512m --executor-cores 2 --queue default --deploy-mode cluster --conf spark.eventLog.enabled=true --conf spark.eventLog.dir=file:///eventLogging <jar\_name> s3a://<bucket\_name>/<file\_name> s3a://<bucket\_name>/output

5. Monitoring the application and check the application results

* \*Monitor Spark job status.

kubectl get endpoints

curl <spark\_master\_pod\_ip>:<port>

* \*Monitor Spark logs for Troubleshooting.

kubectl get pods

kubectl logs -f <spark\_worker\_pod\_name>

kubectl exec -it <spark\_worker\_pod\_name> bash

tail -f <spark\_home\_dir>/logs/<file\_name>.out

tail -f eventLogging/<file\_name\_with\_app\_id>

* \*View standard I/O of the application.

nano <spark\_home\_dir\_inside\_driver\_worker>/work/<driver\_id>/stdout

nano <spark\_home\_dir\_inside\_driver\_worker>/work/<driver\_id>/stderr

nano <spark\_home\_dir\_inside\_worker>/work/<app\_id>/<partition\_number>

/stdout

nano<spark\_home\_dir\_inside\_worker>/work/<app\_id>/<partition\_number>

/stderr

* We can see the progress of our program in the log file of the driver which runs in one of the pods. To get the IP of the driver, check the logs of the master pod. Once the driver IP is determined, use this IP to enter the bash of pod running the driver. Run the following command to check the logs:

tail -f <spark\_home\_dir>/logs/<file\_name>.out

* \*Check health of cluster using the following commands.

kubectl cluster-info

kubectl get nodes

kubectl get pods

kubectl describe pods <pod\_name>

kubectl get deployments

* To check the output folder specified while running the program please run the following command to download the output directory from s3. The aws cli interface is configured with our credentials. They can be replaced.

aws s3 cp s3://spark-cloud/output/ output --recursive

//This will download the output directory to a folder named 'output' in the directory from which this command is executed.

***Summary of steps to test the application on the pi cluster***

**Instructions how to install and run simulator on the Pi cloud :**

The instructions are included in the README of the bitbucket repo as well.

The Pi in the black case is the Kubernetes master. The rest are K8s slaves.  
  
1. Perform the following on Master node.  
 sudo su  
  
 kubeadm reset  
  
 kubeadm init --pod-network-cidr 10.244.0.0/16  
  
 su pirate  
  
 sudo cp /etc/kubernetes/admin.conf $HOME/  
  
 sudo chown $(id -u):$(id -g) $HOME/admin.conf  
   
 export KUBECONFIG=$HOME/admin.conf  
  
 curl -sSL https://rawgit.com/coreos/flannel/v0.7.1/Documentation/kube-flannel-rbac.yml | kubectl create -f -  
  
 curl -sSL https://rawgit.com/coreos/flannel/v0.7.1/Documentation/kube-flannel.yml | sed "s/amd64/arm/g" | kubectl create -f -  
  
 sudo iptables -P FORWARD ACCEPT  
  
2. Perform the following on the slave nodes.  
 sudo su  
  
 kubeadm reset  
  
 <kubeadm\_join\_command\_issued\_by\_master\_node> // once it is initialized it will give such a command for slaves to join.  
  
 sudo iptables -P FORWARD ACCEPT  
  
3. On the master node, do the following to set up the Spark cluster:  
 cd spark  
   
 kubectl create -f spark-master.yaml  
   
 kubectl get pods  
   
 //identify the 'spark-master' pod.  
   
 //enter the shell of the master using:  
   
 kubectl exec -it <name\_of\_master\_pod> bash  
   
 tail -f spark/logs/\_(whatever is the name of the only file here)  
   
 //wait till the screen shows "I HAVE BEEN ELECTED: I AM ALIVE!"  
   
 exit // exits master shell.  
   
 kubectl create -f spark-master-service.yaml  
   
 kubectl create -f spark-worker.yaml  
   
 kubectl get pods  
   
 //identify the 'spark-master' pod.  
   
 //enter the shell of the master using:  
   
 kubectl exec -it <name\_of\_master\_pod> bash  
   
 tail -f spark/logs/\_(whatever is the name of the only file here)  
   
 //wait till the screen shows "Registered worker..."  
   
 spark-submit --properties-file s3.properties --class com.hortonworks.example.Main --master spark://spark-master:7077 --num-executors 4 --driver-memory 1024m --executor-memory 1024m --executor-cores 4 --queue default --deploy-mode cluster --conf spark.eventLog.enabled=true --conf spark.eventLog.dir=file:///eventLogging mc.jar s3a://spark-cloud/input/companies\_list.txt s3a://spark-cloud/input/\*.csv s3a://spark-cloud/output  
   
 //wait till the screen shows "Registering app monte-carlo-var-calculator" and "Launching executor app.. on worker.."  
   
 exit //exit master's shell  
   
 kubectl get pods  
   
 //pick the first worker in the list, copy it's name  
   
 kubectl exec -it <spark\_worker\_name> bash  
   
 cd spark/work  
   
 ls  
   
 //will show a "driver" file directory  
   
 cd "driver.." whatever the name is  
   
 tail -f stdout  
   
//you will see the output of our program here.  
//to check the output folder specified while running the program please run the following command to download the output directory from s3. The aws cli interface is configured with our credentials. The credentials can be replaced with any other user’s credentials.

aws s3 cp s3://spark-cloud/output/ output --recursive

//This will download the output directory to a folder named 'output' in the directory from which this command is executed.

****

From left to right: Ashwani Khemani, Dipankar Ghosh, Gurpreet Kaur Chabada, Mayuri Kumari, Giovanni Agugini Bassi, Pratik Anil Kshirsagar, Riccardo Pressiani.