Kajol Tanesh Shah
A20496724
Fall 2022

CSP554—Big Data Technologies

Assignment #9

Exercise 1) 5 points

Read the article "Real-time stream processing for Big Data" available on the blackboard in the 'Articles' section and then answer the following questions:

a. (1.25 points) What is the Kappa architecture and how does it differ from the lambda architecture?

Ans. In the kappa architecture, everything's a stream. And so, a stream processing engine is used where all the computations are done. What the lambda architecture would call batch processing is simply streaming through historic data. The Kappa architecture uses a powerful stream processor that can deal with data faster than the data that comes in. The kappa architecture represents a swing of the pendulum back to a one-size-fits-all solution. Lambda has both batch-oriented and real-time processing systems. The lambda architecture consists of a batch processing layer (platform) and a transient real-time processing layer (platform), plus a merging layer on top.

b. (1.25 points) What are the advantages and drawbacks of pure streaming versus micro-batch real-time processing systems?

Ans. Purely stream-oriented systems such as Storm and Samza provide very low latency and relatively high per-item cost, while batch-oriented systems achieve unparalleled resource-efficiency at the expense of latency that is prohibitively high for real-time applications. Micro-batch processing systems like Storm Trident and Spark Streaming employ micro-batching strategies to trade latency against throughput thus improving efficiency. Trident groups tuples into batches to relax the one-at-a-time processing model in favor of increased throughput, whereas Spark Streaming restricts batch size in a native batch processor to reduce latency

c. (1.25 points) In few sentences describe the data processing pipeline in Storm.

Ans. A data pipeline or application in Storm is called a topology and as illustrated in Figure 4 is a directed graph that represents data flow as directed edges between nodes which again represent the individual processing steps: The nodes that ingest data and thus initiate the data flow in the topology are called spouts and emit tuples to the nodes downstream which are called bolts and do processing, write data to external storage and may send tuples further downstream themselves. Storm comes with several groupings that control data flow between nodes, e.g. for shuffling or hash-partitioning a stream of tuples by some at- tribute value, but also allows arbitrary custom groupings. By default, Storm distributes spouts and bolts across the nodes in the cluster in a round-robin fashion, though the scheduler is pluggable to

account for scenarios in which a certain processing step has to be executed on a particular node. The application logic is encapsulated in a manual definition of data flow and the spouts and bolts which implement interfaces to define their behavior during start-up and on data ingestion or receiving a tuple, respectively.

d. (1.25 points) How does Spark streaming shift the Spark batch processing approach to work on real-time data streams?

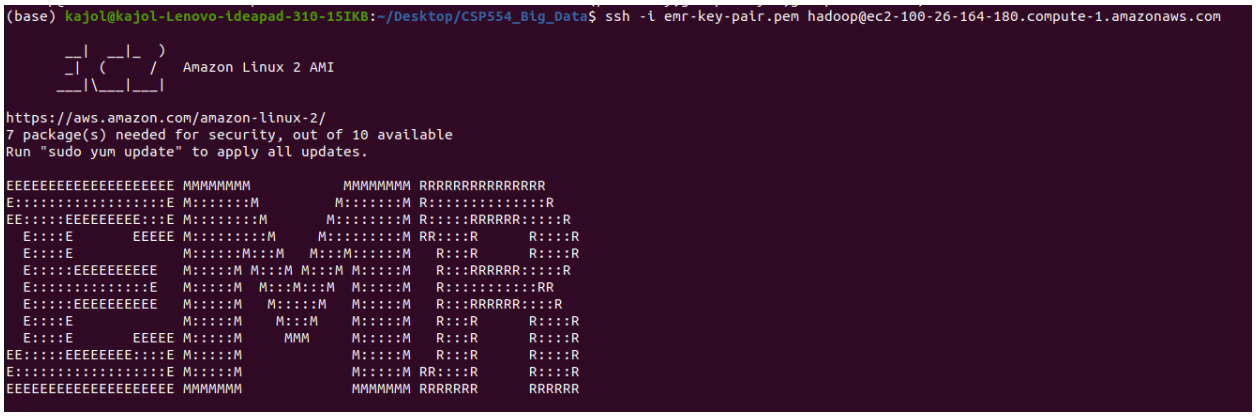
Ans. Spark Streaming shifts Spark's batch-processing approach towards real-time requirements by chunking the stream of incoming data items into small batches, transforming them into RDDs and processing them as usual. It further takes care of data flow and distribution automatically

Exercise 2) 5 points (extra credit; if you don't want to try or if you try and can't get things to work, this won't impact your score negatively)

Step A – Start an EMR cluster

Start up an EMR/Hadoop cluster as previously, but instead of choosing the "Core Hadoop" configuration chose the "Spark" configuration (see below), otherwise proceed as before.

Start up an EMR/Hadoop cluster as previously, but instead of choosing the "Core Hadoop" configuration chose the "Spark" configuration (see below), otherwise proceed as before.



Step B – Copy the Kafka software to the EMR master node

Download the kafka_2.13-3.0.0.tgz file from the blackboard to you PC/MAC. Use the secure copy (scp) program to move this file to the /home/hadoop directory of the master node. Here is an example of how your command line might look (yours will be somewhat different because your master node DNS name, key-pair and kafka file locations will vary):

scp -i ~/emr-key-pair-2.cer /home/kajol/Desktop/CSP554_Big_Data/kafka_2.13-3.0.0.tgz hadoop@ec2-100-26-164-180.compute-1.amazonaws.com:/home/hadoop

```
(base) kajol@kajol-Lenovo-ideapad-310-15IKB:~/Desktop/CSP554_Big_Data$ scp -i
emr-key-pair.pem /home/kajol/Desktop/CSP554_Big_Data/kafka_2.13-3.0.0.tgz hado
op@ec2-100-26-164-180.compute-1.amazonaws.com:/home/hadoop
kafka_2.13-3.0.0.tgz

              41%   35MB   1.6MB/s   00:30 ETA
kafka_2.13-3.0.0.tgz

             100%   82MB   1.4MB/s   01:00
(base) kajol@kajol-Lenovo-ideapad-310-15IKB:~/Desktop/CSP554_Big_Data$ _
```

Step C – Install the Kafka software and start it

Open up a terminal connection to your EMR master node. Over the course of this exercise, you will need to open up three separate terminal connections to your EMR master node. This is the first, which we will call Kafka-Term:

Enter the following command:

tar -xzf kafka_2.13-3.0.0.tgz

Note, this will create a new directory (kafka_2.13-3.0.0) holding the kakfa software release.

Then enter this command:

pip install kafka-python

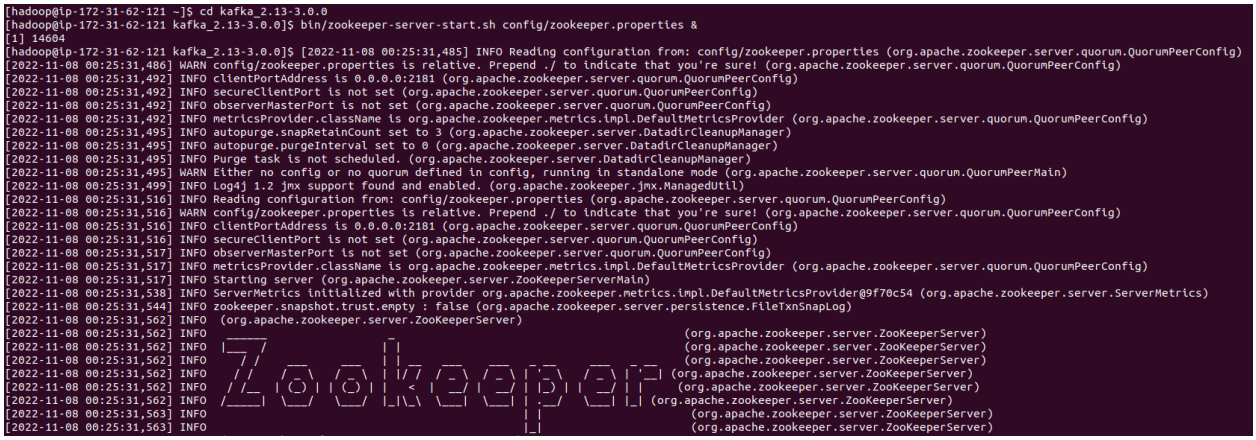
This installs the kafka-python package.

```
[hadoop@ip-172-31-62-121 ~]$ tar -xzf kafka_2.13-3.0.0.tgz
[hadoop@ip-172-31-62-121 ~]$ ls
kafka_2.13-3.0.0  kafka_2.13-3.0.0.tgz
[hadoop@ip-172-31-62-121 ~]$ pip install kafka-python
Defaulting to user installation because normal site-packages is not writeable
Collecting kafka-python
  Downloading kafka_python-2.0.2-py2.py3-none-any.whl (246 kB)
     |                                | 246 kB 33.9 MB/s
Installing collected packages: kafka-python
Successfully installed kafka-python-2.0.2
```

Now enter the following commands into the terminal:

cd kafka_2.13-3.0.0

bin/zookeeper-server-start.sh config/zookeeper.properties &



bin/kafka-server-start.sh config/server.properties &

Step D – Prepare to run Kafka producers and consumers

Open a second terminal connection to the EMR master node. Going forward we will call this terminal connection: Producer-Term.

Open a third terminal connection to the EMR master node. Going forward we will call this terminal connection: Consumer-Term.

Step E – Create a Kafka topic

In the Producer-Term, enter the following command:

cd kafka_2.13-3.0.0

bin/kafka-topics.sh --create --replication-factor 1 --partitions 1 --bootstrap-server localhost:9092 --topic sample

Here we create a new kafka topic called ‘sample’. You can use this command to create a topic with any name you like. Try creating a few more topics.

To list the topics that you created you can enter the following into the Producer-Term (note some default topics already exist):

bin/kafka-topics.sh --list --bootstrap-server localhost:9092

a)

In the Producer-Term (or some other way) write a small program, call it ‘put.py’, using the vi text or some other way of putting a python program onto the EMR master node. If you like you could use a text editor on your PC/MAC to write the program and then scp it over to your EMR master name.

This program should implement a kafka producer that writes three messages to the topic ‘sample’. Recall that you need to convert values and keys to type bytes. The three messages should have keys and values as follows:

Key	Value
‘**MYID**’	Your student id
‘**MYNAME**’	Your name
‘**MYEYECOLOR**’	Your eye color (make it up if you can’t remember)

```
 GNU nano 2.9.8                                                    put.py

from time import sleep
from json import dumps
from kafka import KafkaProducer

Producer = KafkaProducer(bootstrap_servers = ['localhost:9092'],value_serializer = lambda x: dumps(x).encode('utf-8'))

synmyid = 'MYID'
synmyname = 'MYNAME'
synmyeyecolor = 'MYEYECOLOR'

id = input("Enter your ID: ")
name = input("Enter your name: ")
eyecolor = input("Enter your eye color: ")

my_dict = {}
my_dict[synmyid] = id

my_dict1 = {}
my_dict1[synmyname] = name

my_dict2 = {}
my_dict2[synmyeyecolor] = eyecolor

myid = my_dict
Producer.send('sample',myid)
sleep(4)

myname = my_dict1
Producer.send('sample',myname)
sleep(4)

myeyecolor = my_dict2
Producer.send('sample',myeyecolor)
sleep(4)

Producer.close()
```

Execute this program in the Producer-Term, use the command line (you might need to provide a full pathname depending on where your python program is such as /home/hadoop/someplace/put.py):

python put.py

Submit the program as your answer to ‘part a’ of this exercise.

```
[hadoop@ip-172-31-62-121 kafka_2.13-3.0.0]$ nano put.py
[hadoop@ip-172-31-62-121 kafka_2.13-3.0.0]$ python put.py
Enter your ID: A20496724
Enter your name: Kajol Tanesh Shah
Enter your eye color: Brown
[hadoop@ip-172-31-62-121 kafka_2.13-3.0.0]$
```

b)

In the Consumer-Term, write another small program, call it ‘get.py’, using the vi text or some other way of putting a python program onto the EMR master node.

This program should implement a kafka consumer that reads the messages you wrote previously from the topic ‘sample’ and writes them to the terminal.

The output should look something like this:

Key=MYID, Value='your id number'

Key=MYNAME, Value='your name'

Key=MYEYECOLOR, Value='your eye color'

```
 GNU nano 2.9.8                                                    get.py

from ensurepip import bootstrap
from kafka import KafkaConsumer
from json import loads

Consumer = KafkaConsumer('sample', bootstrap_servers = ['localhost:9092'],auto_offset_reset='earliest',enable_auto_commit=True,
group_id='my-group',value_deserializer = lambda x:loads(x.decode('utf-8')))

for i in Consumer:
    for key, value in i.value.items():
        print("key=%s value=%s" % (key,value))

Consumer.close()
```

Execute this program in the Consumer-Term. Use the command line:

python get.py

Note, if needed you can terminate the program by entering 'ctrl-c'.

Submit the program and a screenshot of its output as your answer to 'part b' of this exercise.

```
Last login: Tue Nov  8 00:37:37 2022 from 23-123-17-111.lightspeed.cicril.sbcglobal.net

       __|  __|_  )
       _|  (     /   Amazon Linux 2 AMI
      ___|\___|___|

https://aws.amazon.com/amazon-linux-2/
7 package(s) needed for security, out of 10 available
Run "sudo yum update" to apply all updates.

EEEEEEEEEEEEEEEEEEE MMMMMMMM           MMMMMMMM RRRRRRRRRRRRRRR
E::::::::::::::::::E M:::::::M         M:::::::M R::::::::::::::R
EE:::::EEEEEEEEE:::E M::::::::M       M::::::::M R:::::RRRRRR:::::R
  E::::E       EEEEE M:::::::::M     M:::::::::M RR::::R      R::::R
  E::::E             M::::::M:::M   M:::M::::::M   R:::R      R::::R
  E:::::EEEEEEEEEE   M:::::M M:::M M:::M M:::::M   R:::RRRRRR:::::R
  E::::::::::::::E   M:::::M  M:::M:::M  M:::::M   R::::::::::::RR
  E:::::EEEEEEEEEE   M:::::M   M:::::M   M:::::M   R:::RRRRRR::::R
  E::::E             M:::::M    M:::M    M:::::M   R:::R      R::::R
  E::::E       EEEEE M:::::M     MMM     M:::::M   R:::R      R::::R
EE:::::EEEEEEEE::::E M:::::M             M:::::M   R:::R      R::::R
E::::::::::::::::::E M:::::M             M:::::M RR::::R      R::::R
EEEEEEEEEEEEEEEEEEE MMMMMMM             MMMMMMM RRRRRRR      RRRRRR

[hadoop@ip-172-31-62-121 ~]$ nano get.py
[hadoop@ip-172-31-62-121 ~]$ python get.py
key=MYID value=A20496724
key=MYNAME value=Kajol Tanesh Shah
key=MYEYECOLOR value=Brown

^Z
[1]+  Stopped                 /usr/bin/python3 get.py
```