CSP 554 BIG DATA TECHNOLOGIES
MOHAMMED WASIM R D(A20497053)

ASSIGNMENT 9

Exercise 1)

a) What is the Kappa architecture and how does it differ from the lambda architecture?

The fundamental idea behind the kappa design is to avoid constantly recalculating all data in the batch layer. Instead, all computation is done in the stream processing system, and recalculation only occurs when the business logic changes, which is accomplished by replaying earlier data. By combining a potent stream processor with a scalable streaming system for data preservation, the Kappa Architecture is able to process data considerably more quickly than it is received. For instance, the streaming system Kafka was developed specifically to communicate with the stream processor Samza in this kind of architecture.The effort needed to replay the entire history effect does, however, increase with data volume and the naive approach of retaining the entire change stream may have significantly higher storage requirements than routinely processing new data and updating an existing database. This is because the streaming layer's ability to compress data will determine whether and how well it can be replayed. In cases where unlimited retention times or effective compaction are not necessary, the Kappa Architecture should only be considered as a Lambda Architecture substitute.

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

Although there are some fundamental ideas and operating principles that are shared by all stream processors, there is a substantial variance between the various systems that has a direct impact on processing speed. By contrast, buffering and processing data items in batches increases efficiency while lengthening the time each item spends in the data pipeline. Handling data items as they come in minimizes latency at the expense of high per-item expenses (for example, through message). In contrast to batch-oriented systems, which offer unparalleled resource efficiency at the cost of unacceptably high latency for real-time applications, Storm and Samza are pure stream-oriented systems with very low latency and moderately high per-item costs. Many systems, such Storm Trident and Spark Streaming, use micro-batching techniques to trade latency for throughput since the distance between these two extremes is so great: While Spark Streaming restricts batch size in a native batch processor to reduce latency, Trident aggregates tuples into batches in order to relax the one-at-a-time processing strategy in favor of faster throughput.

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

In Storm, a data pipeline or application is referred to as a topology. As shown, this topology is a directed graph with directed edges between nodes representing the various processing phases as the data flow: Spouts are the nodes in the topology that ingest data and so start the data flow. Spouts emit tuples to bolts, the nodes downstream that perform processing, write data to external storage, and occasionally send tuples farther downstream. Storm includes various built-in groupings to manage data flow across nodes, such as those for

shuffling or hash-partitioning a stream of tuples according to an attribute value, but it also supports arbitrary new groupings. Storm distributes spouts and bolts around the cluster's nodes in a round-robin fashion by default, but the scheduler is pluggable to take into account circumstances in which a specific processing step must be carried out on a specific node, for example due to hardware requirements.

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

By dividing the stream of incoming data items into manageable batches, converting them into RDDs, and processing them as usual, Spark Streaming adapts Spark's batch-processing strategy to real-time requirements. Additionally, it handles data distribution and flow automatically. Since late 2011, Spark Streaming has been under development, and in February 2013, it was added to Spark. Since Spark Streaming is a component of the Spark framework and uses the same API as Spark, both systems have a sizable development community and a sizable pool of potential consumers from the beginning. It can therefore be made robust. Samza and Storm component failures, and it also backs dynamically scaling the resources devoted to a certain application. Intaken data is turned into a series of RDDs. which is first processed through workers and is referred to as DStream (discretized stream). While data objects inside an RDD are handled in parallel without any assurances of ordering, all RDDs in a DStream are processed in order. Batch sizes below 50 mstend to be impractical because there is a slight work scheduling delay involved in processing an RDD. As a result, processing an RDD requires the best case scenario is roughly 100 ms, however Spark Streaming is designed for latency in the range of a few moments. Spark Streaming offers the option of employing a write-ahead log (WAL), from which data can be replayed after failure, to prevent data loss even for unreliable data sources. A state DStream that can be updated by a DStream transformation is used to implement state management.

Exercise 2)

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.

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 /Users/nachdaph/csp554-fall-2021/assignments/kafka_2.13-3.0.0.tgz hadoop@ec2-3-21-286-125.us.east-2.compute.amazonaws.com:/home/hadoop

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

```
Last login: Mon Nov  7 22:53:04 on console
(base) mohammedwasimrd@Mohammeds-Air ~ % cd Downloads
(base) mohammedwasimrd@Mohammeds-Air Downloads % chmod 400 keypair.pem
(base) mohammedwasimrd@Mohammeds-Air Downloads % ssh -i keypair.pem hadoop@ec2-3-21-206-125.us-east-2.compute.amazonaws.com
The authenticity of host 'ec2-3-21-206-125.us-east-2.compute.amazonaws.com (3.21.206.125)' can't be established.
ED25519 key fingerprint is SHA256:ABYpu+2UM9otVcu2vuvRSDCipfPumVvrfOSYDtzXpfk.
This key is not known by any other names
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
Warning: Permanently added 'ec2-3-21-206-125.us-east-2.compute.amazonaws.com' (ED25519) to the list of known hosts.

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

https://aws.amazon.com/amazon-linux-2/
17 package(s) needed for security, out of 20 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-172-31-30-92 ~]$ tar -xvf kafka_2.13-3.0.0.tar
kafka_2.13-3.0.0/
kafka_2.13-3.0.0/LICENSE
kafka_2.13-3.0.0/NOTICE
kafka_2.13-3.0.0/bin/
kafka_2.13-3.0.0/bin/kafka-delete-records.sh
kafka_2.13-3.0.0/bin/trogdor.sh
kafka_2.13-3.0.0/bin/connect-mirror-maker.sh
kafka_2.13-3.0.0/bin/kafka-console-consumer.sh
kafka_2.13-3.0.0/bin/kafka-consumer-perf-test.sh
kafka_2.13-3.0.0/bin/kafka-log-dirs.sh
kafka_2.13-3.0.0/bin/zookeeper-server-stop.sh
kafka_2.13-3.0.0/bin/kafka-verifiable-consumer.sh
kafka_2.13-3.0.0/bin/kafka-features.sh
kafka_2.13-3.0.0/bin/kafka-acls.sh
kafka_2.13-3.0.0/bin/zookeeper-server-start.sh
kafka_2.13-3.0.0/bin/kafka-server-stop.sh
kafka_2.13-3.0.0/bin/kafka-configs.sh
kafka_2.13-3.0.0/bin/kafka-reassign-partitions.sh
kafka_2.13-3.0.0/bin/kafka-leader-election.sh
kafka_2.13-3.0.0/bin/kafka-producer-perf-test.sh
kafka_2.13-3.0.0/bin/kafka-transactions.sh
kafka_2.13-3.0.0/bin/kafka-topics.sh
kafka_2.13-3.0.0/bin/connect-standalone.sh
kafka_2.13-3.0.0/bin/kafka-metadata-shell.sh
kafka_2.13-3.0.0/bin/kafka-get-offsets.sh
kafka_2.13-3.0.0/bin/kafka-dump-log.sh
kafka_2.13-3.0.0/bin/kafka-broker-api-versions.sh
kafka_2.13-3.0.0/bin/kafka-consumer-groups.sh
kafka_2.13-3.0.0/bin/connect-distributed.sh
kafka_2.13-3.0.0/bin/kafka-delegation-tokens.sh
```

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.

Now enter the following commands into the terminal:

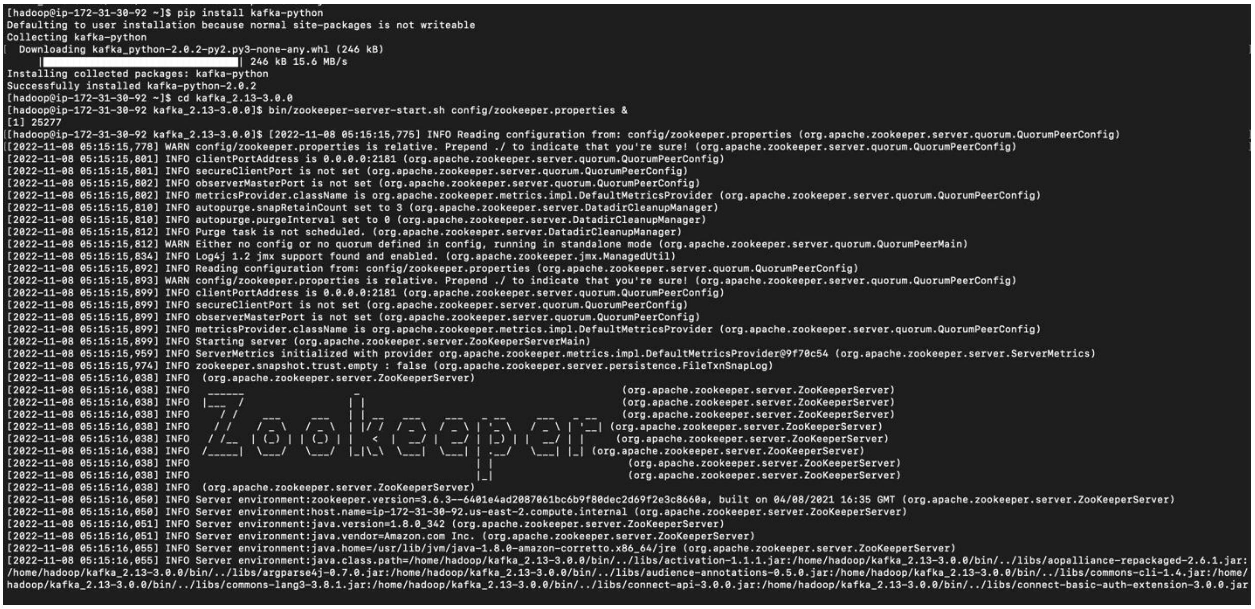
cd kafka_2.13-3.0.0

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

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

Remember to end each line with the ampersand (&). This starts up a zookeeper instance and the kafka server. Lots of messages should appear. You might need to tap the return/enter key after messages appear to see the Linux prompt again.

Just leave this terminal window alone after you enter these commands. As you interact with kafka this terminal will display low level diagnostic messages which you can ignore.



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:

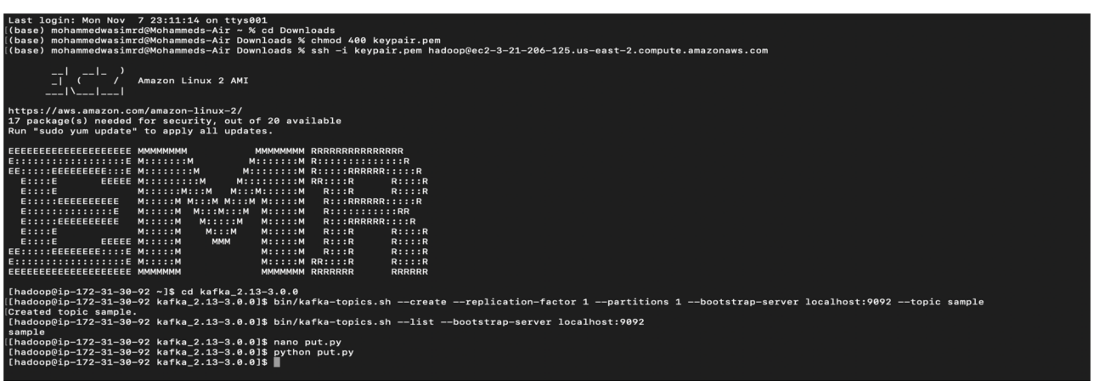
cd kafka_2.13-3.0.0

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)

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):

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

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

producer.send('sample', value = 'A20497053',key = 'MYID')
producer.send('sample', value = 'Mohammed Wasim',key = 'MYNAME')
producer.send('sample', value = 'Black',key = 'MYEYECOLOR')

sleep(5)

producer.close()
```

python put.py

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

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'

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.

```
from kafka import KafkaConsumer
from json import loads

consumer = KafkaConsumer(
'sample', auto_offset_reset='earliest',
bootstrap_servers=['localhost:9092'],
consumer_timeout_ms=1000,value_deserializer=lambda x: loads(x.decode('utf-
8')),key_deserializer=lambda x: loads(x.decode('utf-8')))

for message in consumer:
```