ASSIGNMENT 09

Exercise 1:

a)The essential thought behind the Kappa engineering isn't to recompute all data in the group layer intermittently, however, to perform recomputation just when the business rationale changes and to do all calculation in the stream handling framework alone.The contrast between Kappa architecture over Lambda architecture is its effortlessness. We need to keep two distinct cycles and perhaps two unique arrangements of groups, with Lambda which puts a ton of strain on little financial plan projects. In Kappa, there's just a single degree of cycle and one bunch of groups. So Kappa is less expensive to keep up with. Besides, from the end-client viewpoint, Kappa requires just a single module to peruse the information, though Lambda requires two unique perspectives for batch and real-time data results. And also in the lambda architecture, data is also stored in a persistence layer, such as HDFS, from which it is regularly ingested and processed by the batch layer, while the speed layer handles the portion of the data that has not yet been processed by the batch layer, and the serving layer consolidates both by merging the batch and speed layer's output.

b)Purely stream-oriented systems, such as Storm and Samza, have extremely low latency (advantage) but extremely high per-item costs (disadvantage), whereas batch-oriented systems for real-time applications, achieving unmatched resource efficiency at extremely high latency is prohibitively expensive. Because the difference between these two extremes is so vast, micro-batching algorithms are used by systems like Storm Trident and Spark Streaming to exchange latency for throughput: As a bonus, Trident combines tuples into batches, allowing it to relax the one-at-a-time processing method in favor of improved throughput, whereas Spark Streaming limits batch size in a native batch processor as a disadvantage, in order to reduce latency.

c)A topology in Storm is a directed graph that depicts data flow as directed edges between nodes representing certain processing steps: Spouts are the nodes in the topology that take in input and start the data flow. Tuples are sent to bolts, which process the data, write it to external storage, and maybe transfer it downstream. Storm includes a number of built-in groupings for controlling data flow across nodes, such as shuffling or hash-partitioning a stream of tuples by some attribute value, but it also allows for arbitrary custom groupings. Spouts are the nodes that intake data and start the data flow in the topology. They emit tuples to bolts, which execute processing, write data to external storage, and may transmit tuples further downstream.
Storm distributes spouts and bolts in a round-robin method throughout the cluster's nodes, while the scheduler is pluggable to allow for circumstances in which a specific processing step must be performed on a specific node.
The application functionality is encased in a manual data flow specification, as well as the spouts and bolts, which implement interfaces to define their behavior during startup and when receiving a tuple, respectively.

d)Spark Streaming adapts Spark's batch-processing strategy to real-time requirements by chunking the stream of incoming data items into small batches, transforming them to RDDs, and processing them as usual. It also manages data flow and distribution automatically. Data is imported and converted into a DStream of RDDs before being processed by employees (discretized stream). All RDDs are processed in order in a DStream, whereas data within an RDD is handled in parallel with no guarantee of order. And also it automatically manages data flow and distribution. Spark Streaming can be made resilient to the failure of any component, such as Storm and Samza, because it runs on top of a shared Spark cluster. It also permits dynamically scaling the resources provided to an application.

Extra credits:

a)

```
[[hadoop@ip-172-31-25-132 kafka_2.13-3.0.0]$ bin/kafka-topics.sh --list --bootstrap-server localhost:9092
__consumer_offsets
sample
sample01
sample02
sample03
[[hadoop@ip-172-31-25-132 kafka_2.13-3.0.0]$ cd ..
[[hadoop@ip-172-31-25-132 ~]$ python put4.py
[hadoop@ip-172-31-25-132 ~]$
```

Program : put.py

```
from json import dumps
from kafka import KafkaProducer

producer = KafkaProducer(bootstrap_servers=['localhost:9092'])


r = { "MYID":"A20488730","MYNAME":"Rahul Maddula","MYEYECOLOR":"Balck"}

for i in r.keys():
    sk =bytes(i, 'utf-8')
    pk=bytes(r[i], 'utf-8')
    producer.send('sample',value = sk,key=pk)


producer.close()
```

b)

```
Last login: Mon Mar 28 22:58:59 on ttys002
(base) rahulmaddula@rahuls-Air ~ % cd downloads/
(base) rahulmaddula@rahuls-Air downloads % chmod 400 my-key-pair.pem
(base) rahulmaddula@rahuls-Air downloads % ssh -i my-key-pair.pem hadoop@ec2-18-206-235-90.compute-1.amazonaws.com
Last login: Tue Mar 29 04:21:05 2022

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

https://aws.amazon.com/amazon-linux-2/
1 package(s) needed for security, out of 7 available
Run "sudo yum update" to apply all updates.
-bash: warning: setlocale: LC_CTYPE: cannot change locale (UTF-8): No such file or directory

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-172-31-25-132 ~]$ python get.py
key=MYID value=A20488730
key=MYNAME value=Rahul Maddula
key=MYEYECOLOR value=Balck
[hadoop@ip-172-31-25-132 ~]$
```

Program get.py

```
from kafka import KafkaConsumer

consumer = KafkaConsumer(
   'sample',
    bootstrap_servers=['localhost:9092'],
    auto_offset_reset='earliest',
    consumer_timeout_ms=10000,
    group_id='my-group')

for message in consumer:
   print ("key=%s value=%s" % (message.key.decode('utf-8'), message.value.decode('utf-8')))

consumer.close()
```