CSP554—Big Data Technologies

Prachi Kotadia(A20549927)

Assignment #9

Readings

Read (From the Free Books and Chapters section of our blackboard site):

To learn more about Apache Kafka:

- Kafka: The Definitive Guide (Chapter 1, and then further for more in depth information only as you are interested)

To lean about Spark Structured Streaming:

- Learning Spark, Ch. 8 (pp. 207-234)
- Spark: The Definitive Guide (pp.26-32)

Worth: 5 points + 5 points extra credit

Due by the start of the next class period

Assignments should be uploaded via the Blackboard portal

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?**

Kappa Architecture vs. Lambda Architecture:

Kappa Architecture is designed to simplify the data processing pipeline by having a single stream processing layer handle all data processing tasks. It does this by eliminating the batch layer found in Lambda Architecture, relying instead on a powerful stream processing system and a scalable streaming system for data retention, like Kafka. The aim is to process data in real-time, only reverting to historical data replay for cases like business logic changes. This approach seeks simplicity but may face challenges with large data volumes and storage requirements.

Lambda Architecture, on the other hand, includes both a batch processing layer and a stream (speed) processing layer. It aims to provide a comprehensive data processing model that caters to both real-time and large-scale batch processing needs, thus handling both the Volume and the Velocity aspects of Big Data. This dual-layered approach offers flexibility and reliability but at the cost of increased complexity in development, deployment, and maintenance

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

Pure streaming systems, such as Storm and Samza, offer very low latency processing but at a relatively higher per-item cost due to the overhead of handling each item individually.

Micro-batch systems, exemplified by Storm Trident and Spark Streaming, attempt to balance throughput and latency by processing small batches of data. This approach can still achieve low latency (millisecond order) while improving throughput compared to pure streaming systems. The trade-off involves a slight increase in latency due to the batching process but significantly improved resource efficiency compared to one-at-a-time streaming

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

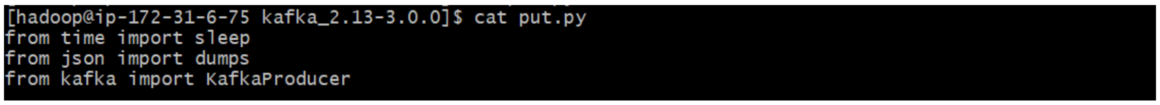
In Storm, the data processing pipeline is organized into a topology, a directed graph where nodes represent processing steps (spouts and bolts) and directed edges represent the flow of data. Spouts ingest data into the topology, and bolts process the data, possibly emitting new tuples to other bolts for further processing. This design allows for flexible, scalable, and fault-tolerant real-time data processing. Storm's architecture supports at-least-once processing semantics through tuple acknowledgement, ensuring no data loss but at the cost of increased messaging overhead.

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

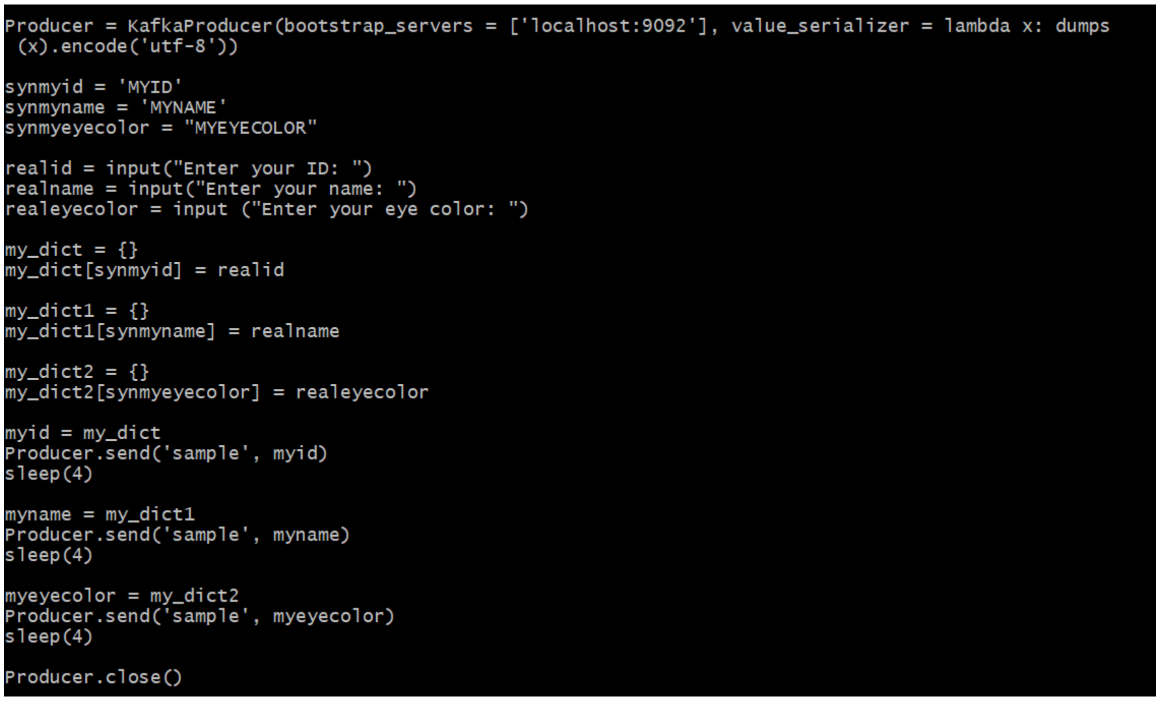
Spark Streaming extends the Spark batch processing model to handle real-time data streams by dividing the data stream into small batches. These batches are then processed by the Spark engine as mini-batch jobs, creating a series of resilient distributed datasets (RDDs) for each batch. This approach allows Spark to apply its fast, in-memory batch processing capabilities to streams of data, achieving near real-time processing. Spark Streaming integrates seamlessly with Spark's batch processing and machine learning libraries, offering fault tolerance, scalability, and the convenience of developing batch and streaming applications using the same API.

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)

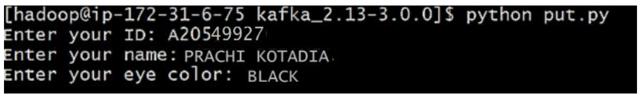
a) Producer Terminal:



```
[hadoop@ip-172-31-6-75 kafka_2.13-3.0.0]$ cat put.py
from time import sleep
from json import dumps
from kafka import KafkaProducer
```

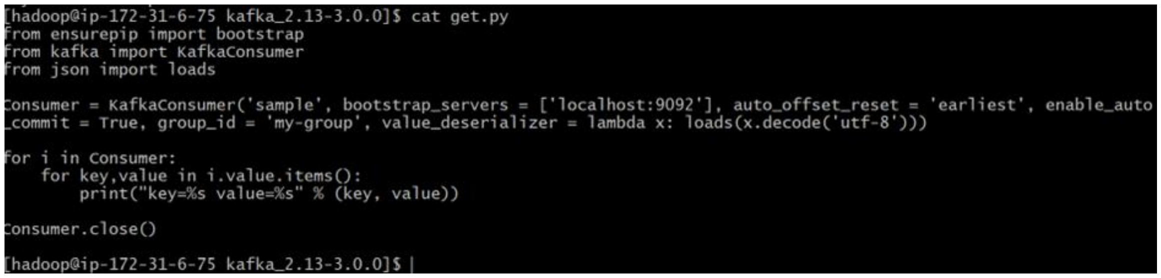


```
Producer = KafkaProducer(bootstrap_servers = ['localhost:9092'], value_serializer = lambda x: dumps
 (x).encode('utf-8'))

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

realid = input("Enter your ID: ")
realname = input("Enter your name: ")
realeyecolor = input ("Enter your eye color: ")

my_dict = {}
my_dict[synmyid] = realid

my_dict1 = {}
my_dict1[synmyname] = realname

my_dict2 = {}
my_dict2[synmyeyecolor] = realeyecolor

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()
```

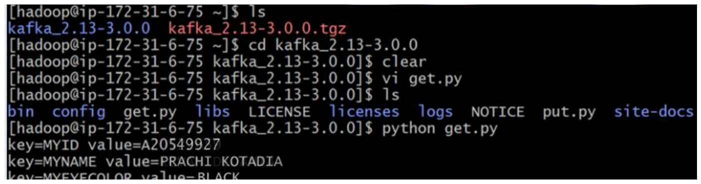


```
[hadoop@ip-172-31-6-75 kafka_2.13-3.0.0]$ python put.py
Enter your ID: A20549927
Enter your name: PRACHI KOTADIA
Enter your eye color:  BLACK
```

b) Consumer Terminal:



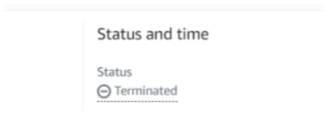
```
[hadoop@ip-172-31-6-75 kafka_2.13-3.0.0]$ cat get.py
from ensurepip import bootstrap
from kafka import KafkaConsumer
from json import loads

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()

[hadoop@ip-172-31-6-75 kafka_2.13-3.0.0]$ |
```



```
[hadoop@ip-172-31-6-75 ~]$ ls
kafka_2.13-3.0.0  kafka_2.13-3.0.0.tgz
[hadoop@ip-172-31-6-75 ~]$ cd kafka_2.13-3.0.0
[hadoop@ip-172-31-6-75 kafka_2.13-3.0.0]$ clear
[hadoop@ip-172-31-6-75 kafka_2.13-3.0.0]$ vi get.py
[hadoop@ip-172-31-6-75 kafka_2.13-3.0.0]$ ls
bin  config  get.py  libs  LICENSE  licenses  logs  NOTICE  put.py  site-docs
[hadoop@ip-172-31-6-75 kafka_2.13-3.0.0]$ python get.py
key=MYID value=A20549927
key=MYNAME value=PRACHI KOTADIA
key=MYEYECOLOR value=BLACK
```

c)

Remember to terminate your EMR cluster!



Status and time

Status
Terminated