

Exercise 1)

Read the article “The Lambda and the Kappa” found on our blackboard site in the “Articles” section and answer the following questions using between 1-3 sentences each. Note this, article provides a real-world and critical view of the lambda pattern and some related big data processing patterns:

1. Extract-transform-load (ETL) is the process of taking transactional business data (think of data collected about the purchases you make at a grocery store) and converting that data into a format more appropriate for reporting or analytic exploration. What problems was encountering with the ETL process at Twitter (and more generally) that impacted data analytics?

Answer –

- ELT pipelines were difficult to build.
 - ELT pipelines were difficult to implement.
 - ELT pipelines introduced latency (data processing was done on the previous day's data).
 - The requirement for updated data in real time was not met.
2. What example is mentioned about Twitter of a case where the lambda architecture would be appropriate?

Answer –

The example mentioned about Twitter of a case where the lambda architecture would be appropriate is counting tweet impressions using real-time and history data of a specific message.

3. What did Twitter find were the two of the limitations of using the lambda architecture?

Answer –

The two limitations the Twitter found about using the lambda architecture are:

- Complexity cost-
Everything must be done twice, once for batch processing and once for streaming. Both must be completed in parallel, which frequently necessitates the collaboration of two teams.
 - Uncertain computing semantics-
Unpredictable fluctuations in aggregate values due to missing data in real time as a result of some loss during stream processing, but these losses are reflected in batch processing afterwards.
4. What is the Kappa architecture?

Answer –

The Kappa architecture is based on historical data stream processing. In this architecture everything is just a stream, so it needs an processing engine.

5. Apache Beam is one framework that implements a kappa architecture. What is one of the distinguishing features of Apache Beam?

Answer –

Apache Beam presents a rich API that explicitly recognizes the difference between event time, the time when an event actually occurred, and processing time, the time when the event is observed in the system. which aids with stream processing with only one engine.

Exercise 2)

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

Answer –

One use of the Kappa architecture is in streaming data processing. The core idea underlying Kappa architecture is that a single technological stack can be used for both batch and real-time processing, which is very helpful for analytics.

Additionally, the Kappa architecture facilitates historical analytics, which subsequently reads the batch-stored streaming data from the messaging engine to generate more analysable outputs for a wider range of analyses. Real-time analytics are also made possible by the Kappa design since data is read and transformed as soon as it is fed into the messaging system.

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

Answer –

Furthermore, historical analytics are made easier by the Kappa architecture, which then reads the batch-stored streaming data from the messaging engine to produce additional outputs that can be analysed for a larger variety of studies. Since data is read and modified as soon as it is supplied into the messaging system, the Kappa architecture also makes real-time analytics possible.

The world has evolved, and micro batches are insufficient for many purposes. Micro-batch processing is frequently used by organisations to make design choices that defy stream processing. For instance, Spark streaming, which is simply an extension of the Spark API for micro-batch processing with the added benefit of using in-memory computing resources, may be used in an Apache Spark shop. Systems are able to react to fresh data events as they happen thanks to stream processing.

Stream processing systems capture and process data as it is created, as opposed to gathering and organising it at predetermined intervals.

Micro-batch processing has worked well in some situations, but businesses are quickly realising that stream processing based on in-memory technology—whether on-premises or in the cloud—is the superior option. Technology for stream processing is being used in

current applications quite quickly. Businesses have shifted to real-time analysis in response to the explosion of data over the past ten years in order to address a variety of use cases and applications, responding to data closer to the moment of generation.

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

Answer –

The Zookeeper cluster is what the Apache Storm is searching for. A storm cluster consists of Storm Nimbus and Storm Supervisor, at least two different components. Hadoop Job Tracker has a near master node in Storm Nimbus.

It is in charge of allocating responsibilities to subordinates, spreading technology throughout the cluster, and making sure errors are not made. Storm Supervisor is one of the office's nodes. It runs the code that Storm Nimbus gives it. Failed-safe and stateless are storm clusters. A Storm cluster can function without a Nimbus node since Nimbus distributes the code to supervisors if there is a Zookeeper and at least one supervisor node. In addition, Storm offers Storm UI, Storm Log reader, and Storm DRPC.

The concept of a job—a batch operation with a final outcome and end—is used by Hadoop and other systems of a similar nature to process data. Hurricane uses topologies, which are execution graphs where each node has processing logic and each edge specifies how data is routed among nodes.

A storm topology consists of two streams: bolts and spouts. A spout serves as a representation of the source stream. An agent for processing stream data that can create additional sources is called a bolt. For example, a bolt might be used to consume and output a stream of hot topics from a spout that was connected to the Twitter API and produced a tweet stream.

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

Answer –

By splitting up incoming data into smaller batches, converting them to RDDs, and processing them as usual, Spark Streaming adapts Spark's batch-processing approach to real-time requirements. Additionally, data transmission and flow are automated. Data is imported and transformed into a DStream of RDDs (discretized stream) before being processed by personnel. While data within an RDD is treated in parallel without any guarantee of the order in a DStream, all RDDs are processed in the order.

Exercise 3)

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):

```
python put.py
```

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

Answer –

```
from json import dumps
from kafka import KafkaProducer
producer = KafkaProducer(bootstrap_servers=['localhost:9092'])
p = {
    "MYID": "A20551908",
    "MYNAME": "Vadlamudi Manogna",
    "MYEYECOLOR": "Brown"
}
for k in p.keys():
    bkeys = bytes(k, 'utf-8')
    bvalue = bytes(p[k], 'utf-8')
    producer.send('sample', value = bvalue, key = bkeys)
producer.close()
~
~
```

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:

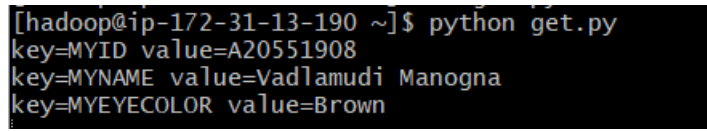
python get.py

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.

Answer:

Output:



```
[hadoop@ip-172-31-13-190 ~]$ python get.py
key=MYID value=A20551908
key=MYNAME value=Vadlamudi Manogna
key=MYEYECOLOR value=Brown
```

Program:

```
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()
```

Exercise 4)

These steps illustrate how to execute a pyspark (Python) spark streaming job. The job accepts a sequence of lines that the user types in onto one terminal window over a 10 second interval and then counts the number of distinct words in those lines and outputs the word count results to a second terminal window. This continues every 10 seconds. To do this we will set up a Spark EMR cluster and connect two terminal windows to it. In the first we will run the Linux 'nc' (Netcat) command. It will open a TCP socket on port 3333.

After it does so, any line you then type will be sent out on that port. In another terminal window we will execute a pyspark word count program that will set up the spark streaming pipeline using DStreams. Our initial DStream will be connected to and read the lines from port 3333 and then go on to perform the word count process. So on one terminal (connected to the EMR master node) you might see:

```
[hadoop@ip-172-31-19-223 ~]$ nc -lk ec2-3-91-10-18.compute-1.amazonaws.com 3333
```

this is a test of the the system <- note

Answer –

Output of the word count program running on EC-1 terminal is:

Sentence: this is a test of the system

```
-----
Time: 2023-11-17 03:47:10
-----
('this', 1)
('is', 1)
('test', 1)
('of', 1)
('a', 1)
('the', 1)
('system', 1)
```

Sentence: this is a test of the the system

```
-----
Time: 2023-11-22 23:51:00
-----
('this', 1)
('is', 1)
('test', 1)
('of', 1)
('a', 1)
('the', 2)
('system', 1)
```