CSP554—Big Data Technologies

Assignment #8

Worth: 20 points

Exercise 1) 5 points

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?

ANS:
ETL pipeline construction and maintenance were challenging. They created latency, which led to the usage of day-old data for business intelligence and data analytics. Organisations needed more up-to-date, real-time data to make decisions as business grew faster. This led to an increase in ETL frequency, but it also put more strain on the already shoddy ETL system, sometimes causing it to break.

2. What example is mentioned about Twitter of a case where the lambda architecture would be appropriate?

ANS: The platforms for batch processing and transient real-time processing, as well as the merging layer on top, make up the lambda architecture.
The example mentioned about the Twitter case is - Let's say that we wanted to track tweet impressions. As users are already touching, swiping, and clicking, we also want real-time updates as they happen, as well as historical counts going all the way back to the moment a tweet was posted. In Twitter, impressions data was captured by frontend logs.
MapReduce served as the batch processing layer in Twitter's lambda architecture, where aggregations were commonplace. The logging pipeline did, however, add a delay; even in the best scenario, logs were a few hours old. The real-time layer enters at this point, in this case being Storm, a real-time processing framework that offers an abstraction of a dataflow graph (vertices represent calculations, while edges capture data movement).
Real-time aggregations were carried out by Storm, and the logic for fusing batch and real-time data was contained in a merging layer. Findings from the real-time layer might be discarded once the (delayed) results from the batch layer were received.

In other words, real-time results were temporary, whereas batch computations gave the truth.

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

ANS: - First Limitation is the Complexity.

The lambda design requires that everything be written twice: once for the batch platform and again for the real-time platform. Two separate implementations must always be maintained in parallel, often by different teams. That is, for the results to be dependable, the modifications have to be transferred from one to the other. Even in cases where the lambda architecture operates as planned, the semantics of the calculations remain unclear. To provide a specific example, aggregate values may occasionally fluctuate suddenly. Let's say that for ten minutes, there was a temporary spike in stress on the Storm cluster, resulting in the loss of log data. This would remain unknown to everyone until the logs are processed by the batch layer at a later time.

Because persistence is an explicit design goal, logging pipelines typically follow a separate code route than the real-time processing layer and are therefore more reliable. In this case, the missing data reappear, and the total numbers abruptly alter.

The second limitation is expressiveness (no arbitrary computations). Around the end of 2012, Twitter started working on the next iteration of their lambda architecture, which would eventually become Summingbird. Summingbird is a domain specific language (DSL) that facilitates the automatic translation of queries into Storm topologies or MapReduce tasks. The key finding of Summingbird is that certain algebraic structures provide the theoretical foundation for combining batch and online processing in an elegant way. This leads to a major simplification in practise: all aggregations must be stated as commutative monoids.

4. What is the Kappa architecture?

ANS: - In the Kappa architecture, everything is a stream. If everything is a stream, all you need is a stream processing engine. A log is an append-only collection, and so a stream can be thought of as such. All that a table has is a cache of the most recent value for each key in the log, and the log contains a history of all table modifications. Kafka is essentially a distributed log.

An attempt is made to get back to a one-size-fits-all strategy using the kappa architecture.

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

ANS: Apache Beam has a rich API that clearly differentiates between event time—the actual time an event occurred—and processing time—the moment the system becomes aware of the event. For example, an event occurring at 4:17 (event time) may not be visible until 4:20 (processing time) due to logging pipeline delays.

The purpose of a watermark is to demonstrate the relationship between the two and to guarantee that the observed data is accurate in terms of the times of the events. For example, a heuristic watermark could record the fact that 99 percent of messages will arrive within 5 minutes of the event time. Watermarks cause moving window aggregations, therefore in our case, computing the aggregation would make sense.

The Apache Beam API offers one potential abstraction for early triggering. The Beam idea does not distinguish between batch and streaming calculations. The primary distinction is actually between constrained and unbounded datasets. Generally speaking, batch processing is just streaming across bounded datasets. This is a basic example of a kappa architecture.

Exercise 2)

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

ANS: Kappa Architecture's fundamental idea is to only do recomputation when the business logic changes by replaying historical data, as opposed to performing all computation in the batch layer on a regular basis.

The Kappa Architecture does this by using a strong stream processor that can handle data at a pace many times faster than it is coming in and a scalable streaming system for data retention. Kafka, which was created expressly to interact with the stream processor Samza in this kind of architecture, is an illustration of such a streaming system.

Simply applications that don't require unbounded retention times or that allow for effective compaction (such as those where it makes sense to only keep the most recent value for each) can be thought of as a viable alternative to the Lambda Architecture.

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

ANS:
Purely stream-oriented systems 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.
- Systems like Storm Trident and Spark Streaming employ micro-batching strategies to trade latency against
throughput: 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.
- We employ stream processing when we need to analyze or provide data as soon as we can after we obtain it because we process data as soon as it enters the storage layer, which is frequently also extremely near to the time it was generated.
- When a significant volume of data suddenly surfaces in stream processing and needs to be handled, the system might quickly become overwhelmed if there are insufficient storage and processing capacity.
- Micro-batch processing is helpful when we require extremely recent data but not necessarily in real-time. In other words, we don't have the time to wait for a batch processing to complete in an hour or a day, but we also don't need to know what just transpired in the past few seconds.
- When done correctly, Micro-batch processing can be quite effective, but because of its size, large-scale failures are more likely to happen.

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

ANS:

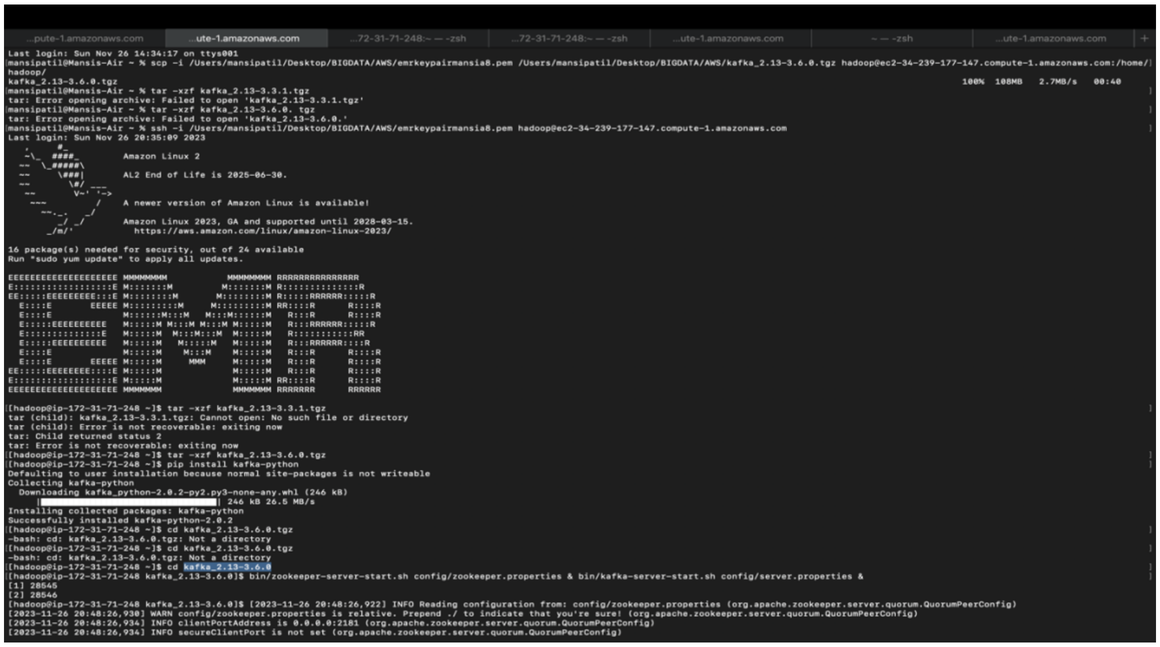
- Similar to the batch processing abstraction provided by the MapReduce paradigm, its programming model gave a stream processing abstraction.
Storm is fault-tolerant, scalable, and elastic because to its ability to redistribute work while it's operating. Storm provides reliable state implementations that are recoverable and resistant to supervisor failure. After being ingested from the streaming layer, data is shared throughout Storm components before the final output reaches the serving layer.
In Storm, a data pipeline or application is referred to as a topology. It is a directed graph that depicts data flow as directed edges between nodes, where nodes once more stand in for the various processing steps: Spouts, or the nodes that intake input and therefore start the data flow in the topology, emit tuples to bolts, or the nodes downstream, which do processing, write data to external storage, and may even send tuples themselves. Several groupings that regulate data flow between nodes are provided by Storm, such as those for shuffle or hash-partitioning a stream of tuples by an attribute value, but it also permits arbitrary custom groupings.

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

ANS:

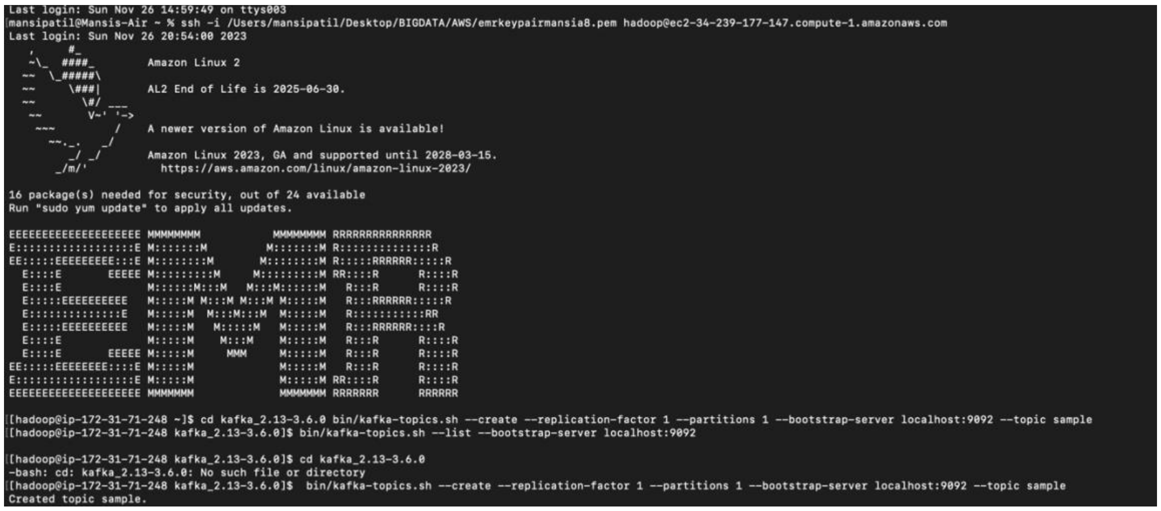
- By breaking up the stream of incoming data items into smaller batches, converting them into RDDs, and processing them as usual, Spark Streaming shifts Spark's batch-processing technique to real-time requirements. Additionally, it handles data distribution and flow automatically. Spark Streaming provides dynamically scaling the resources allotted for an application and may be made resilient to failure of any component, such as Storm and Samza. Before processing through workers, data is ingested and turned into a series of RDDs known as a DStream (discretized stream). While data items inside an RDD are handled in parallel without any assurances of ordering, all RDDs in DStream are processed in order. Batch sizes lower than 50 ms are frequently impractical because of the job scheduling delay that occurs when processing an RDD.
A state DStream that can be updated by a DStream transformation is used to implement state management

Exercise 3) 5 points



```
Last login: Sun Nov 26 14:34:17 on ttys001
mansipatil@Mansis-Air ~ % scp -i /Users/mansipatil/Desktop/BIGDATA/AWS/emrkeypairmansia8.pem /Users/mansipatil/Desktop/BIGDATA/AWS/kafka_2.13-3.6.0.tgz hadoop@ec2-34-239-177-147.compute-1.amazonaws.com:/home/
hadoop/
kafka_2.13-3.6.0.tgz                                                                                        100%  108MB   2.7MB/s   00:40
mansipatil@Mansis-Air ~ % tar -xzf kafka_2.13-3.3.1.tgz
tar: Error opening archive: Failed to open 'kafka_2.13-3.3.1.tgz'
mansipatil@Mansis-Air ~ % tar -xzf kafka_2.13-3.6.0. tgz
tar: Error opening archive: Failed to open 'kafka_2.13-3.6.0.'
mansipatil@Mansis-Air ~ % ssh -i /Users/mansipatil/Desktop/BIGDATA/AWS/emrkeypairmansia8.pem hadoop@ec2-34-239-177-147.compute-1.amazonaws.com
Last login: Sun Nov 26 20:35:09 2023
   ,     #_
   ~\_  ####_        Amazon Linux 2
  ~~  \_#####\
  ~~     \###|       AL2 End of Life is 2025-06-30.
  ~~       \#/ ___
   ~~       V~' '->
    ~~~         /    A newer version of Amazon Linux is available!
      ~~._.   _/
         _/ _/       Amazon Linux 2023, GA and supported until 2028-03-15.
       _/m/'           https://aws.amazon.com/linux/amazon-linux-2023/

16 package(s) needed for security, out of 24 available
Run "sudo yum update" to apply all updates.

EEEEEEEEEEEEEEEEEEEE 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
EEEEEEEEEEEEEEEEEEEE MMMMMMM             MMMMMMM RRRRRRR      RRRRRR

[hadoop@ip-172-31-71-248 ~]$ tar -xzf kafka_2.13-3.3.1.tgz
tar (child): kafka_2.13-3.3.1.tgz: Cannot open: No such file or directory
tar (child): Error is not recoverable: exiting now
tar: Child returned status 2
tar: Error is not recoverable: exiting now
[hadoop@ip-172-31-71-248 ~]$ tar -xzf kafka_2.13-3.6.0.tgz
[hadoop@ip-172-31-71-248 ~]$ 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 26.5 MB/s
Installing collected packages: kafka-python
Successfully installed kafka-python-2.0.2
[hadoop@ip-172-31-71-248 ~]$ cd kafka_2.13-3.6.0.tgz
-bash: cd: kafka_2.13-3.6.0.tgz: Not a directory
[hadoop@ip-172-31-71-248 ~]$ cd kafka_2.13-3.6.0.tgz
-bash: cd: kafka_2.13-3.6.0.tgz: Not a directory
[hadoop@ip-172-31-71-248 ~]$ cd kafka_2.13-3.6.0
[hadoop@ip-172-31-71-248 kafka_2.13-3.6.0]$ bin/zookeeper-server-start.sh config/zookeeper.properties & bin/kafka-server-start.sh config/server.properties &
[1] 28545
[2] 28546
[hadoop@ip-172-31-71-248 kafka_2.13-3.6.0]$ [2023-11-26 20:48:26,922] INFO Reading configuration from: config/zookeeper.properties (org.apache.zookeeper.server.quorum.QuorumPeerConfig)
[2023-11-26 20:48:26,930] WARN config/zookeeper.properties is relative. Prepend ./ to indicate that you're sure! (org.apache.zookeeper.server.quorum.QuorumPeerConfig)
[2023-11-26 20:48:26,934] INFO clientPortAddress is 0.0.0.0:2181 (org.apache.zookeeper.server.quorum.QuorumPeerConfig)
[2023-11-26 20:48:26,934] INFO secureClientPort is not set (org.apache.zookeeper.server.quorum.QuorumPeerConfig)
```

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



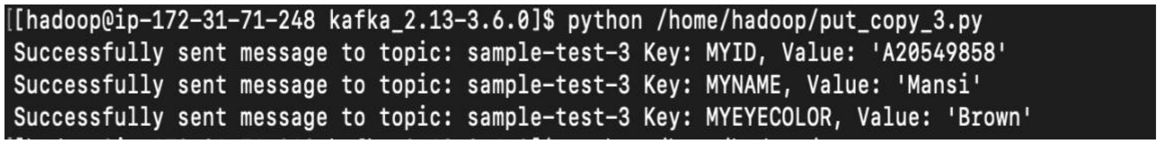
```
Last login: Sun Nov 26 14:59:49 on ttys003
mansipatil@Mansis-Air ~ % ssh -i /Users/mansipatil/Desktop/BIGDATA/AWS/emrkeypairmansia8.pem hadoop@ec2-34-239-177-147.compute-1.amazonaws.com
Last login: Sun Nov 26 20:54:00 2023
   ,     #_
   ~\_  ####_        Amazon Linux 2
  ~~  \_#####\
  ~~     \###|       AL2 End of Life is 2025-06-30.
  ~~       \#/ ___
   ~~       V~' '->
    ~~~         /    A newer version of Amazon Linux is available!
      ~~._.   _/
         _/ _/       Amazon Linux 2023, GA and supported until 2028-03-15.
       _/m/'           https://aws.amazon.com/linux/amazon-linux-2023/

16 package(s) needed for security, out of 24 available
Run "sudo yum update" to apply all updates.

EEEEEEEEEEEEEEEEEEEE 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
EEEEEEEEEEEEEEEEEEEE MMMMMMM             MMMMMMM RRRRRRR      RRRRRR

[hadoop@ip-172-31-71-248 ~]$ cd kafka_2.13-3.6.0 bin/kafka-topics.sh --create --replication-factor 1 --partitions 1 --bootstrap-server localhost:9092 --topic sample
[hadoop@ip-172-31-71-248 kafka_2.13-3.6.0]$ bin/kafka-topics.sh --list --bootstrap-server localhost:9092

[hadoop@ip-172-31-71-248 kafka_2.13-3.6.0]$ cd kafka_2.13-3.6.0
-bash: cd: kafka_2.13-3.6.0: No such file or directory
[hadoop@ip-172-31-71-248 kafka_2.13-3.6.0]$  bin/kafka-topics.sh --create --replication-factor 1 --partitions 1 --bootstrap-server localhost:9092 --topic sample
Created topic sample.
```

Program:

```
 from kafka import KafkaProducer
bootstrap_servers = ['localhost:9092']
topicName = 'sample'
producer = KafkaProducer(bootstrap_servers = bootstrap_servers)
key_bytes = bytes('MYID', encoding='utf-8'
value_bytes = bytes('A20549858', encoding='utf-8')
ack= producer.send(topicName,key=key_bytes,value=value_bytes)
metadata = ack.get()
```

```
print("Successfully sent message to topic: "+metadata.topic+" Key: MYID, Value:
'A20549858'")
key_bytes = bytes('MYNAME', encoding='utf-8')
value_bytes = bytes('Mansi', encoding='utf-8')
ack = producer.send(topicName, key=key_bytes, value=value_bytes)
metadata = ack.get()
print("Successfully sent message to topic: "+metadata.topic+" Key: MYNAME, Value:
'Mansi'")
key_bytes = bytes('MYEYECOLOR', encoding='utf-8')
value_bytes = bytes('Brown', encoding='utf-8')
ack = producer.send(topicName, key=key_bytes, value=value_bytes)
metadata = ack.get()
print("Successfully sent message to topic: "+metadata.topic+"Key: MYEYECOLOR, Value:
'Brown'")
```

Output of Execution:



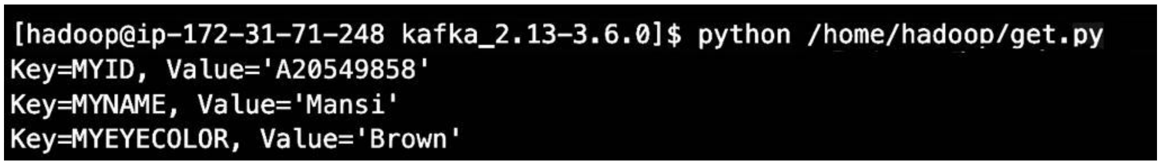
```
[[hadoop@ip-172-31-71-248 kafka_2.13-3.6.0]$ python /home/hadoop/put_copy_3.py
 Successfully sent message to topic: sample-test-3 Key: MYID, Value: 'A20549858'
 Successfully sent message to topic: sample-test-3 Key: MYNAME, Value: 'Mansi'
 Successfully sent message to topic: sample-test-3 Key: MYEYECOLOR, Value: 'Brown'
```

b) Submit the program and a screenshot of its output as your answer to ‘part b’ of this exercise

Program:

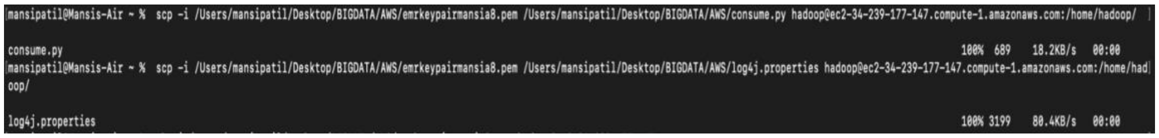
```
from kafka import KafkaConsumer
import sys
bootstrap_servers = ['localhost:9092']
topicName = 'sample'
consumer = KafkaConsumer(topicName,bootstrap_servers =
bootstrap_servers, auto_offset_reset =
'earliest')
try:
        for message in consumer:
            key = message.key.decode("utf-8")
            value = message.value.decode("utf-8")
             print("Key="+key+", Value='"+value+"'")
except KeyboardInterrupt:
             sys.exit()
```

Output of Execution-

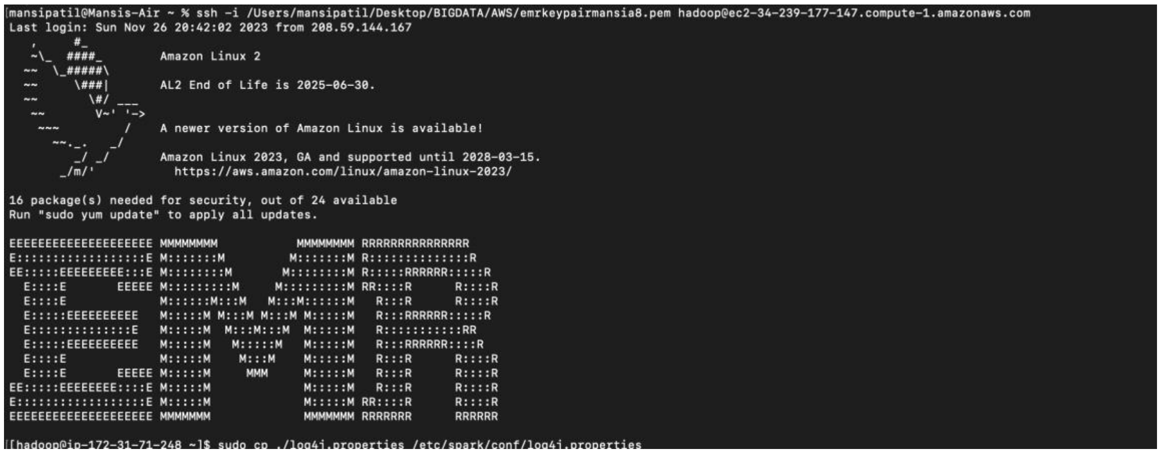

```
[hadoop@ip-172-31-71-248 kafka_2.13-3.6.0]$ python /home/hadoop/get.py
Key=MYID, Value='A20549858'
Key=MYNAME, Value='Mansi'
Key=MYEYECOLOR, Value='Brown'
```

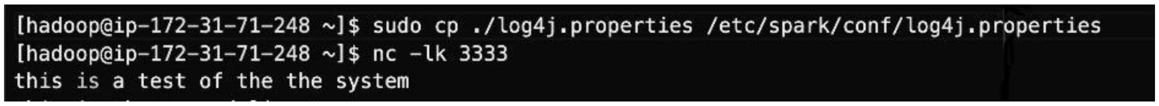
Exercise 4) 5 points

SCP commands –

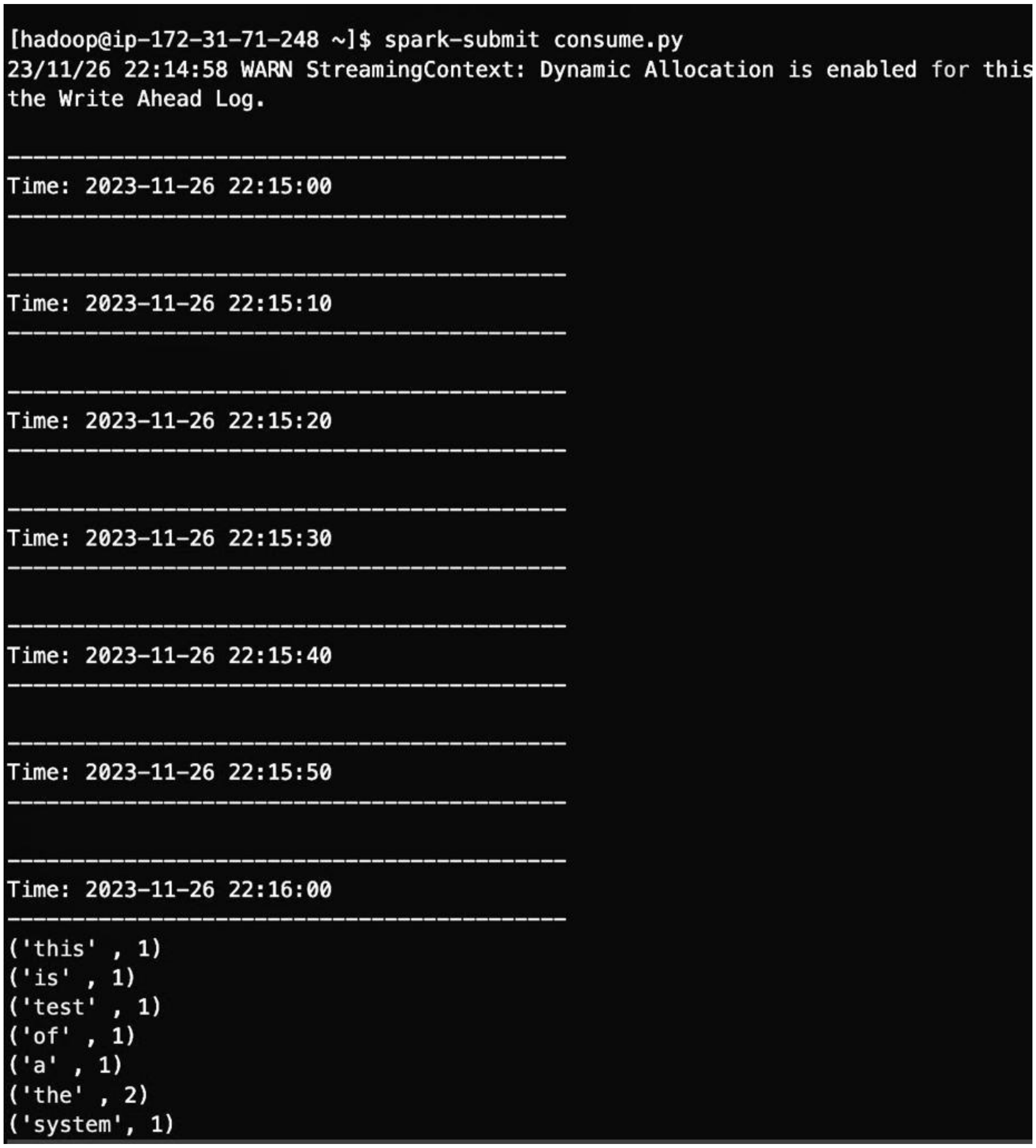

```
mansipatil@Mansis-Air ~ %  scp -i /Users/mansipatil/Desktop/BIGDATA/AWS/emrkeypairmansia8.pem /Users/mansipatil/Desktop/BIGDATA/AWS/consume.py hadoop@ec2-34-239-177-147.compute-1.amazonaws.com:/home/hadoop/

consume.py                                                                                                100%  689    18.2KB/s   00:00
mansipatil@Mansis-Air ~ %  scp -i /Users/mansipatil/Desktop/BIGDATA/AWS/emrkeypairmansia8.pem /Users/mansipatil/Desktop/BIGDATA/AWS/log4j.properties hadoop@ec2-34-239-177-147.compute-1.amazonaws.com:/home/had
oop/

log4j.properties                                                                                          100% 3199    80.4KB/s   00:00
```

EC2- WINDOW 1 SCREENSHOT:


```
mansipatil@Mansis-Air ~ % ssh -i /Users/mansipatil/Desktop/BIGDATA/AWS/emrkeypairmansia8.pem hadoop@ec2-34-239-177-147.compute-1.amazonaws.com
Last login: Sun Nov 26 20:42:02 2023 from 208.59.144.167
   ,     #_
   ~\_  ####_        Amazon Linux 2
  ~~  \_#####\
  ~~     \###|       AL2 End of Life is 2025-06-30.
  ~~       \#/ ___
   ~~       V~' '->
    ~~~         /    A newer version of Amazon Linux is available!
      ~~._.   _/
         _/ _/       Amazon Linux 2023, GA and supported until 2028-03-15.
       _/m/'           https://aws.amazon.com/linux/amazon-linux-2023/

16 package(s) needed for security, out of 24 available
Run "sudo yum update" to apply all updates.

EEEEEEEEEEEEEEEEEEEE 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
EEEEEEEEEEEEEEEEEEEE MMMMMMM             MMMMMMM RRRRRRR      RRRRRR

[hadoop@ip-172-31-71-248 ~]$ sudo cp ./log4j.properties /etc/spark/conf/log4j.properties
```


```
[hadoop@ip-172-31-71-248 ~]$ sudo cp ./log4j.properties /etc/spark/conf/log4j.properties
[hadoop@ip-172-31-71-248 ~]$ nc -lk 3333
this is a test of the the system
```

EC2-2 Window Screenshot:



```
[hadoop@ip-172-31-71-248 ~]$ spark-submit consume.py
23/11/26 22:14:58 WARN StreamingContext: Dynamic Allocation is enabled for this
the Write Ahead Log.

-------------------------------------------
Time: 2023-11-26 22:15:00
-------------------------------------------

-------------------------------------------
Time: 2023-11-26 22:15:10
-------------------------------------------

-------------------------------------------
Time: 2023-11-26 22:15:20
-------------------------------------------

-------------------------------------------
Time: 2023-11-26 22:15:30
-------------------------------------------

-------------------------------------------
Time: 2023-11-26 22:15:40
-------------------------------------------

-------------------------------------------
Time: 2023-11-26 22:15:50
-------------------------------------------

-------------------------------------------
Time: 2023-11-26 22:16:00
-------------------------------------------
('this' , 1)
('is' , 1)
('test' , 1)
('of' , 1)
('a' , 1)
('the' , 2)
('system', 1)
```