Big Data Technologies - CSP 554
Assignment 8 - Apache Spark Streaming
Prabhu Avula | A20522815
Illinois Institute of Technology, Chicago

1. Kafka:

Part A:

```
prabhuavula7@a20522815-n2-csp554-m:~/kafka_2.13-3.0.0$ python put.py
Messages sent successfully.
prabhuavula7@a20522815-n2-csp554-m:~/kafka_2.13-3.0.0$
```

Users > prabhuavula7 > Desktop > A9 > put.py > ...

```python
from kafka import KafkaProducer

bootstrap_servers = ['localhost:9092']

producer = KafkaProducer(bootstrap_servers=bootstrap_servers)

messages = [
    (b'MYID', b'A20522815'),
    (b'MYNAME', b'Prabhu Avula'),
    (b'MYEYECOLOR', b'brown')
]

for key, value in messages:
    producer.send('sample', key=key, value=value)

producer.flush()
producer.close()

print("Messages sent successfully.")
```

Part B:

```
prabhuavula7@a20522815-n2-csp554-m:~$ cd kafka_2.13-3.0.0
prabhuavula7@a20522815-n2-csp554-m:~/kafka_2.13-3.0.0$ python get.py
Key=MYID, Value=A20522815
```

Users > prabhuavula7 > Desktop > A9 > get.py > ...

```python
from kafka import KafkaConsumer
bootstrap_servers = ['localhost:9092']
topic = 'sa
consumer = KafkaConsumer(
    topic,
    bootstrap_servers=bootstrap_servers,
    group_id='my-group',
    auto_offset_reset='earliest'
)

try:
    for message in consumer:
        key = message.key.decode('utf-8') if message.key else None
        value = message.value.decode('utf-8') if message.value else None
        print(f"Key={key}, Value={value}")
        break
finally:
    consumer.close()

```

(import) KafkaConsumer: Any

2. Spark Streaming Demo:

For some reason I could never get it to work as expected. I tried troubleshooting it. I was unable to see the DNS name on the dataproc cluster details page so I tried to go through the console. I found two pieces of relevant information (The natIP in the second screenshot and using that a name in the first screenshot). Tried to execute the subsequent steps using both these in the consume.py program but regardless I couldn't get it to work. I've provided the spark submit job screenshot below as well.

```
7
8   # Create a DStream
9   lines = ssc.socketTextStream("83.148.155.104.bc.googleusercontent.com", 3333)
0
```

```
 8   # Create a DStream
 9   lines = ssc.socketTextStream("104.155.148.83", 3333)
10
```

```
Last login: Mon Jul 22 14:57:13 on ttys009
^[[A%                                                                    (base) prabhuavula7@Prabhus-MacBook-Pro ~ % gcloud com
pute scp /Users/prabhuavula7/Desktop/A9/consume2.py a20522815-n2-csp554-m:/home/prabhuavula7 --zone us-central1-a

consume2.py                                   100%  688     6.2KB/s   00:00
(base) prabhuavula7@Prabhus-MacBook-Pro ~ % gcloud compute ssh a20522815-n2-csp554-m
No zone specified. Using zone [us-central1-a] for instance: [a20522815-n2-csp554-m].
Existing host keys found in /Users/prabhuavula7/.ssh/google_compute_known_hosts
Linux a20522815-n2-csp554-m 6.1.0-22-cloud-amd64 #1 SMP PREEMPT_DYNAMIC Debian 6.1.94-1 (2024-06-21) x86_64

The programs included with the Debian GNU/Linux system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
Last login: Mon Jul 22 20:38:04 2024 from 96.18.234.8
prabhuavula7@a20522815-n2-csp554-m:~$ sudo cp./log4j.properties etc/spark/conf/log4j.properties
sudo: cp./log4j.properties: command not found
prabhuavula7@a20522815-n2-csp554-m:~$ sudo cp ./log4j.properties etc/spark/conf/log4j.properties
cp: cannot create regular file 'etc/spark/conf/log4j.properties': No such file or directory
prabhuavula7@a20522815-n2-csp554-m:~$ ls etc/spark/conf/log4j.properties
ls: cannot access 'etc/spark/conf/log4j.properties': No such file or directory
prabhuavula7@a20522815-n2-csp554-m:~$ ls ./log4j.properties
./log4j.properties
prabhuavula7@a20522815-n2-csp554-m:~$ sudo cp ./log4j.properties /etc/spark/conf/log4j.properties
prabhuavula7@a20522815-n2-csp554-m:~$ ls /etc/spark/conf/log4j.properties
/etc/spark/conf/log4j.properties
prabhuavula7@a20522815-n2-csp554-m:~$ nc -lk 3333
spark-submit consume2.py
```

3. Summary of the Article

The article explores the feasibility of two big data technologies, AsterixDB and Spark Streaming, which are integrated with Cassandra for processing streaming data. Companies like Facebook, LinkedIn, and Twitter analyze streaming data for real-time insights and service development. The study aims to compare these technologies in handling semi-structured data in a simulated social media scenario.

Streaming data analysis is crucial for real-time insights. Various systems, such as Spark, AsterixDB, and Cassandra, have been developed. These systems must be evaluated for performance and scalability in handling large volumes of data. The primary objective is to conduct a feasibility analysis of AsterixDB and Spark Streaming with Cassandra for stream-based processing. The focus is on throughput, latency, and scalability in a distributed cloud environment. The study employs a simulated Twitter-like scenario using Eucalyptus cloud computing on a DSM multiprocessor platform. AsterixDB and Spark Streaming are tested for their real-time ability to process and analyze tweets.

AsterixDB outperforms Spark Streaming with Cassandra regarding throughput and latency, primarily when data feeds are used, and processing is implemented in Java. AsterixDB also scales better or on par with Spark Streaming as the number of nodes in the cloud platform increases. However, AsterixDB experiences delays due to data batching when tweets are streamed into the database.

AsterixDB uses REST API and data feeds for streaming and supports AQL and user-defined functions (UDFs) in various programming languages. It follows a master-slave cluster model and handles semi-structured data. On the other hand, Spark Streaming integrates with multiple data sources like Kafka, Flume, and Twitter. It supports Scala, Java, Python, and R. Spark uses resilient distributed datasets (RDD) for parallel processing and can connect to databases like Cassandra for persistence.

AsterixDB performs better in handling streaming data with higher throughput and lower latency. It also scales effectively in a cloud environment. However, its performance can be hindered by batching delays. While versatile and capable, Spark Streaming is less efficient in comparison, especially in scenarios requiring high throughput and low latency.

The study's findings are significant for organizations evaluating big data technologies for real-time data processing. AsterixDB proves to be a strong candidate for high-performance stream processing tasks, while Spark Streaming remains a viable option for more flexible and integrated data processing solutions. Future research could explore further optimization of AsterixDB to reduce batching delays and expand the comparative analysis to include other emerging stream processing technologies.