**CSP 554 – Assignment #9**

Name: Adarsh Mathad Vijayakumar

CWID: A20424847

Email ID: avijayakumar@hawk.iit.edu

Exercise 1) Read and provide a half page summary and analysis of this article available on the blackboard in the ‘Articles’ section: Beyond Batch Processing: Towards Real-Time and Streaming Big Data

In this article, the author has discussed two categories of non-batch workloads solutions: real-time

processing, and stream processing of big data and for each category, he has discussed paradigms,

strengths and differences to Hadoop and also introduced some practical systems and frameworks for

each category.

* The article talks about Hadoop’s strengths like easy programming model, near-linear speedup

and scalability, and fault tolerance and short coming like total batch processing behavior and

that MapReduce is not able to execute recursive or iterative jobs inherently.

* It also talks about other MapReduce extensions which improve its usability and performance

like Twister and HaLoop, Tez which provides easy API to Yarn, FlumeJava library which

provides data pipelines on top of MapReduce, cascading an abstraction that allows complex

workflows on Hadoop and Phoenix and Metis are two MapReduce frameworks that are designed

for execution on shared memory parallel systems.

* The author then talks about Real-Time Big Data Processing and the two major solutions: in-

memory computing, and real-time queries over big data. In-memory computing uses a distributed

memory storage that can be used either as a standalone input source or as a caching layer for

disk-based storages. In particular, when the input totally fits in distributed memory or when the

job has multiple iterations over input, in-memory computing can significantly reduce execution

time. Solutions to real-time querying over big data mostly use custom storage formats and well-

known techniques from parallel DBMSs to join and aggregation, and hence can respond to

queries in less than a few seconds.

* In the stream-processing sector, there are two popular frameworks: Storm, and S4. Each one has

its own programming model, strengths and weaknesses. Author discussed both frameworks and

their superiority to MapReduce-based systems for stream processing.

* The author believes that solutions to batch and high throughput processing of big data, like

Hadoop, have reached to an acceptable maturity level. However, they are not suitable enough for

non-batch requirements. Considering high demands for interactive queries and big data streams,

in-memory computing stands out as a notable solution that can handle both real-time and stream

requirements. Among discussed frameworks, Spark is a good example for this case which

supports in-memory computing using RDDs, real-time and interactive querying using Shark, and

stream processing using fast micro-batching. However, the future will tell which approach will be

popular in practice.

Exercise 2) Read and provide a half page summary and analysis of this article available on the blackboard in the ‘Articles’ section: Real-time stream processing for Big Data

In this article author provides an overview over some of the most popular distributed stream processing

systems currently available and highlight similarities, differences and trade-offs taken in their

respective designs.

* Section 2 covers the environment in which the processing systems featured in this article are

typically deployed, while the systems focused on are described in Section 3. And later the author

give an overview over other systems for stream processing in Section 4 and conclude in Section 5.

* Batch-oriented systems have done the heavy lifting in data-intensive applications for decades, but

they do not reflect the unbounded and continuous nature of data as it is produced in many real-

world applications. Stream oriented systems, on the other hand, process data as it arrives and thus

are oftentimes a more natural fit, though inferior with respect to efficiency.

* While a growing number of production deployments implementing the Lambda Architecture and

emerging hybrid systems like Dataflow/Beam, Flink or Apex document significant efforts to close

the gap between batch and stream-processing in both research and practice, the Kappa Architecture

completely eschews the traditional approach and even heralds the advent of purely stream-oriented

Big Data analytics. However, whether at the core of novel system designs or as a complement to

existing architectures, author states that horizontally scalable stream processors are gaining

momentum as the requirement for low latency has become a driving force in modern Big Data

analytics pipelines.

Exercise 3) Extra credit

1. Get the spark streaming demo code to work in your Hortonworks sandbox. Provide screen shots of output for various inputs

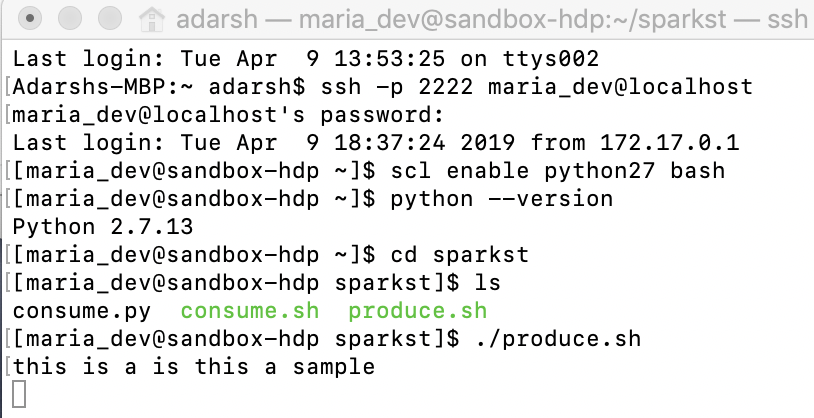
Running the spark streaming demo code in the hortonworks sandbox. Two terminals are opened.

In terminal 1 ./produce.sh was executed and in terminal 2 ./consume.sh was executed.

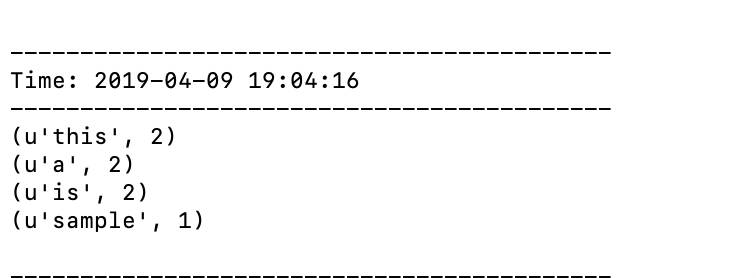
Screenshots of various inputs:

Output 1)

In terminal 1 a sequence of words is given as input. The screenshot of this shown below:

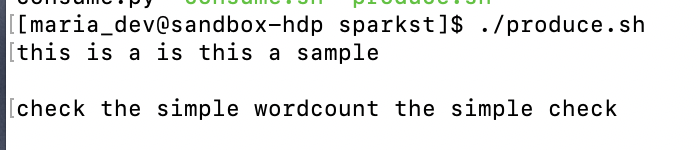


Here in terminal 1 were ./produce.sh is executed, the sequence of words “this is a is this a sample” is given as input. The terminal 2 where ./consume.sh is executed, the output is seen i.e.. the count of number of each word in the given input word sequence. The terminal 2 output screenshot is shown below:

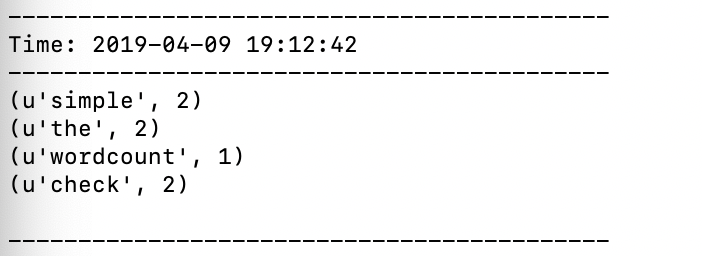


Output 2)

Terminal 1 – input

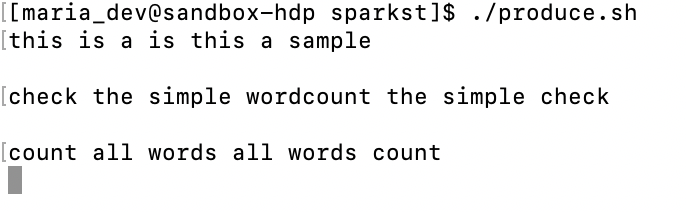


Terminal 2 – output

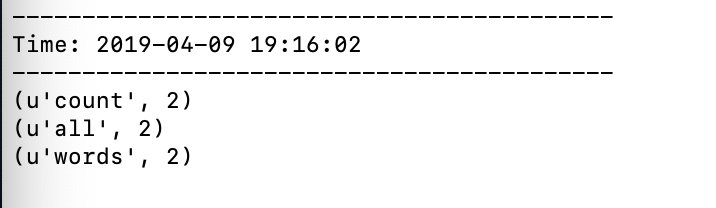


Output 3)

Terminal 1 – Input



Terminal 2 – output



1. Modify consume.py to output a count of words beginning with only the letters a through h inclusive using just RDD transformations and actions. Provide screen shots of output for various inputs.

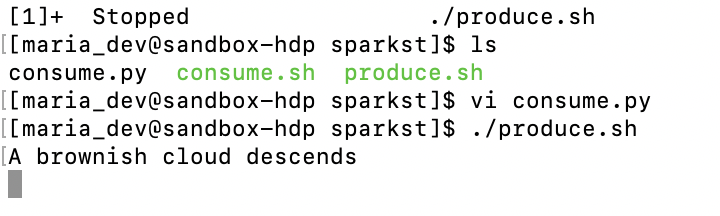
The consume.py is modified to count only the words that begin with a through h. The modified consume.py is shown below:



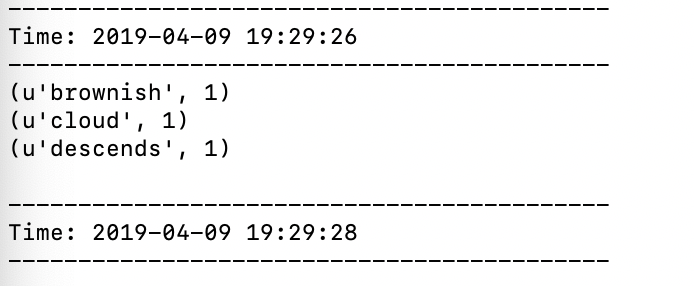
Screenshots of various inputs and their corresponding outputs for the modified consume.py:

Output 1)

Terminal 1 – Input

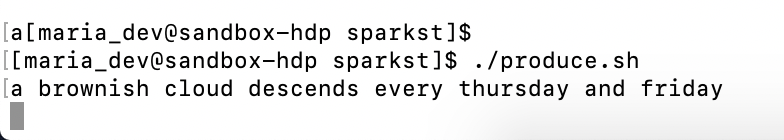


Terminal 2 – output

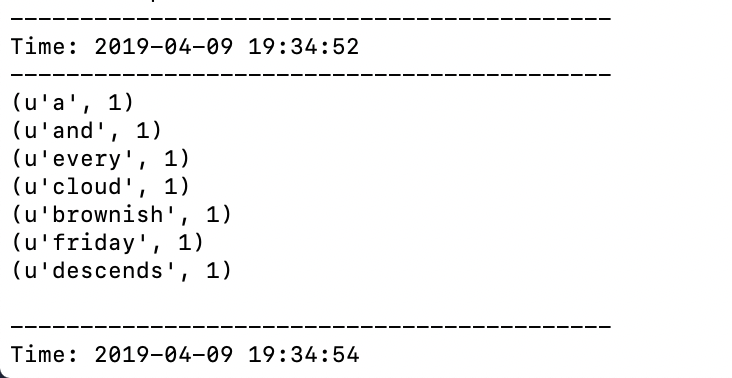


Output 2)

Terminal 1 – Input

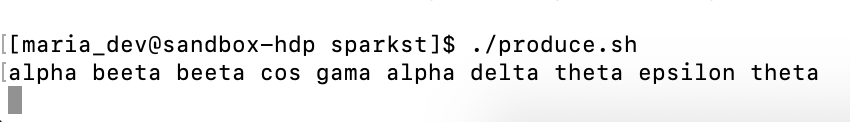


Terminal 2 – output



Output 3)

Terminal 1 – Input



Terminal 2 – output

