Supercomputing for Big Data ET4310 (2016)

Assignment 1 (Using Spark for In-Memory Computation)

Parag Bhosale(4517415)

September 2016

Introduction

The goal of the assignment is to learn basics of Apache Spark which is an in-memory framework for cluster computing which makes it suitable for big data analysis. This assignment is divided into three parts. The first part is to extract some required data from a flat file. The second part is the analysis of live twitter stream. Last part is the further analysis of the data which is stored during live twitter analysis. Background section describes what Apache Spark is about and discusses about bash/stream processing. Various components of spark like RDD, Dstream are stated in this section. In next Implementation section, algorithms of all exercises are discussed briefly. Results are shown in the next section and further conclusions are made in the last section of the report.

Background

Apache Spark

Apache Spark is a powerful open source processing engine built around speed, ease of use, and sophisticated analytics. It was originally developed at UC Berkeley in 2009[1]. It is in-memory computing programming model which keeps **data in memory(RAM)** so that it can exploit **faster access** of the data. This is different from Hadoop which keeps data in the disk. So, in-memory computing makes spark faster than Hadoop. Some of the literature on the internet shows that spark is 10 times faster than the Hadoop. This feature also makes spark able to analyze a live stream of data like Twitter, kinesis or Flume. So Spark is suitable for the application like live data processing from sensors in a factory or online product recommendation or real time marking campaigns.

Spark does not have its own file system. It must use file systems like HDFS. It has only processing component. On the other hand, Hadoop has HDFS and **mapreduce** as the processing component. One can use Hadoop without the spark. But to use spark, one must have storage component like HDFS.

Hadoop is suitable for bash processing and spark is suitable for bash as well as for stream processing. In bash processing, analysis/computation is done on whole data, for example on a file or on a

table in the database. On the other hand, stream processing is done on small real time dataset or on a window of recent data. Computation time for this must be near real time. One can conclude that our first and third exercise in this lab are bash processing exercises while the second exercise is an example of the stream processing.

Resilient Distributed Datasets (RDDs)

Spark revolves around the concept of a resilient distributed dataset (RDD), which is a fault-tolerant collection of read only elements that can be operated on in parallel[2]. These elements are partitioned across the nodes of the cluster. RDD addresses fault-tolerance by **lineage**
The RDD map1 just keeps a reference to its parent and that's a lineage. Spark transformations like map are called lazy transformations as they are not computed right away. They are computed only when an action is required. This lineage can be displayed using **toDebugString()**. This RDD lineage will be used to recompute the data if some failure occurs. RDDs are read-only. It means new RDD must be created for each transformation on an existing RDD.

Discretized Streams (DStreams)

Discretized Stream or DStream is the basic abstraction provided by Spark Streaming[3]. It is a continuous stream of the data generating from a source or a from a transformation of another DStream. Dstream is a continuous set of RDD. Each RDD stores the data for a particular interval of the time. This interval is specified during creation of streaming context.

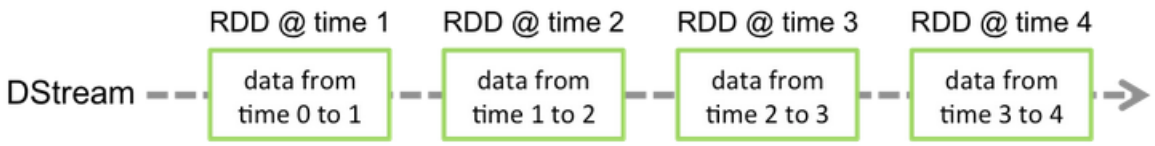

Figure 1: Dstrem

As shown in figure 1, data for the interval from 0 to 1 will be stored in RDD@time 1 and so on. Dstream is a collection of such RDDs. Here Dstream is generated for the continous stream from an input source. Similarly, such DStream can also be generated by transforming (map or flatmap) input Dstream. For example applying flatmap to base Dstream to get words will generate new Dstream.

Implementation

Exercise-1

The goal of the exercise is to learn basic bash processing on a flat file. From an input data, it is required to find out a specific information about views of the pages. From the output of this

exercise, one can find out which language is most viewed. Also which page of that language has most views. To get the desired output we need to filter out unwanted data where language code is same as the page title. This can be achieved using filter transformation or by adding if condition in reduce function. Using if condition in the reduce transformation will generate results faster in case of huge amount of the data than using filter on each line.
Further, we need to apply **mapreduce** component on the data in order to get total views of a language, most viewed page from that language. To achieve this, we need to use the **language code as the key** and remaining column as the value. Finally, we need to sort the output (**sortby** on **TotalViewsInThatLang**)in descending order.

Exercise-2

The second exercise is based on stream processing of Twitter data. In this exercise, we need to find out retweet count of the retweeted tweets. For this we need to filter data using **tweeter4j status** component **isretweet** which provides boolean output true for the retweeted tweets. In this exercise, we are using reduceByKeyAndWindow which has addition parameters like interval and window. These both are time parameters. Window and interval of 60 and 5 seconds mean the data of recent 60 second window will be reduced at interval of 5 seconds.
First, we need to find out retweet count of the original tweet. For this we need to use **getRetweetedStatus().getId()** to get original id of the tweet. Once id is known, we need to find out minimum and maximum of the retweetcount of that particular tweetid. For this **tweetid** is used as key and other relevant fields as value. Further, we need to do $Maximum - minimum + 1$ to get retweetcount of that tweet during that window. The second set of transformation is applied on the previous transformed RDD to find out total retweet count of a particular language. In this we need to use **language code as key** and retweet count as value.
Once these two transformations are done, we need to **join** them with the **language code as the key**. In addition, Apache Tika libray is used to get language code from tweet text.
In the output,we can observe that for each window data is written. Same tweets can appear in the different window with different retweet count means there are recent retweets about that tweet in other intervals.

Exercise-3

In this exercise, further analysis on the data from the second exercise is done. We can say that if we ran the second example for an hour. This exercise does analysis on data of that particular hour. The third exercise is very similar to the second exercise. The main difference we did analysis on an interval of 5 seconds in the previous exercise and now we are doing bash processing on 1-hour data. The two transformations mentioned in the previous section are applied in the same way in this exercise as well.

Results

All three exercises are run on the machine with following specifications.

- Lenovo z51 intel core i5 2.20 GHz GB RAM
- Number of $CPU = 4$ and Number of cores per $socket = 2$

Exercise-1

From table 1, the execution time for this exercise can be observed. Output file part1.txt is generated. Data in the output file is logged as expected in descending order with the highest viewed language on top with most viewed page of that language.

Exercise-2

In this exercise, the programme is coded to wait for one as our window size is one minute. After that for each 5 seconds of batch interval time, the output of the script is loaded in the output file. By observing the output file, we can conclude that for each 5 seconds, the data is loaded with the language having the highest number of tweets in that duration.

Exercise-3

For this exercises, output of second exercise is used as input. Output data is as expected. Run time is observed for the file part2.txt which was provided.

Exercise	Runtime(sec)	File name	Size of the file
exercise 1	7	pagecounts-20160801-040000	330.2 MB
exercise 3	4	part2.txt	22.2 MB

Table 1: Average runtime for the exercises

Conclusion

We have used Apache spark for bash and stream processing. Different transformations and actions are applied on the RDDs and Dstreams. Various applications in both processing can be implemented using Apache spark. For example, data from distributed sensors are streamed continuously to in a factory which is similar to Internet of Things. This run-time analysis can be use to control some actions from the machines. This is related to the second exercise. Further, the data analyst from the same company wants to use data which is logged over the time period of a month to do the analysis. This is similar to exercise 3 and exercise 1. In this way, Apache spark can be used in different scenarios. Also by observing file size and runtime, we can conclude that for bigger size of the data throughput of Spark is higher.

References

- https://databricks.com/spark/about

- http://spark.apache.org/docs/latest/programming-guide.html#resilient-distributed-datasets-rdds
- http://spark.apache.org/docs/latest/streaming-programming-guide.html