Supercomputing for Big Data ET4310 (2016)

Assignment 1 (Using Spark for In-Memory Computation)

Ajaya Adhikari (4627199)

Introduction

This assignment contains exercises to learn the basics about Spark. In exercises 1 and 3 data exploration problems are solved. This gives you a deeper understanding of using map and reduce tasks. Exercise 2 is about Spark streaming using Twitter. Tweets are read continuously every 5 seconds and data is collected in a sliding window of 60 seconds. This report is structured as follows: Introduction, Background, Implementation, Results, Conclusion and References.

Background

RDD

A Resilient Distributed Dataset (RDD) forms the basic data structure in Spark. It is an immutable, distributed and fault tolerant collection of objects that can be operated on in parallel [6].

DStream

A Discretized Stream (DStream) forms the basic abstraction in Spark Streaming. It is a continuous sequence of RDDs which represents a continuous stream of data. It can be created from live data or generated by transformation existing DStreams using operations such as map, window and reduceByKeyAndWindow [7].

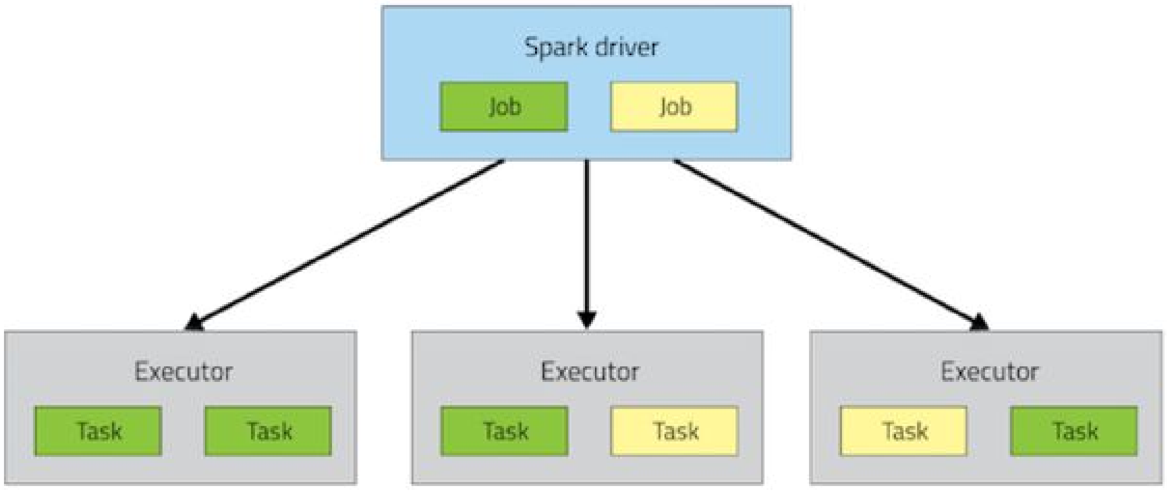
Batch and stream processing

Batch processing is defined as a series of jobs that are connected to each other or executed one after another in sequence or in parallel. It has access to all data and might compute something complex. It is generally more concerned with throughput than latency of individual components of the computation [8].

Stream processing is real-time data processing which receives data that is under constant change. The latency of the processing is very important. The response time should be instantaneous. It computes continuously a function on data elements within a predefined window of recent data [8].

Spark execution

A Spark application consists of a driver process which is in charge of running the user's main function and executing various processes (executor) scattered across nodes on the cluster. The driver process is in charge of the high-level control flow of work, and delegates work to the executor processes. [5]



All transformations in Spark are lazy. They do not compute their results immediately. They just remember the transformations applied to some base data dataset. The transformations are only computed when an action requires a result to be returned to the driver program. Each transformed RDD has to be recomputed every time an action is run on it. If a rdd is used for multiple actions then you can persist the RDD in memory. This will enable faster access the next you query it. [5]

Jobs form the top of the execution hierarchy. Invoking an action inside a Spark application triggers the launch of a Spark job. Spark examines the graph of RDDs on which that action depends and formulates an execution plan. The plan starts from the first RDD which does not depend on another RDD. The execution plan assembles the job’s transformation into stages. Each stage contains a sequence of transformations that can be completed without shuffling the full data. [4]

Spark and Hadoop MapReduce

Spark became popular because it could process big data 100 times faster than Hadoop. But one should take this result with a grain of salt.

Apache Spark processes data in-memory while MapReduce persists to the disk after each map and reduce action. This makes Spark a lot faster than MapReduce. But the downside is that Spark needs a lot of memory. Spark thrives at iterative computations that need to pass over the same data many times. But for one-pass jobs like ETL, the performance of both are comparable [1].

Spark has nice APIs for Java, Scala and Python and it also includes Spark SQL. It is also possible to write user defined functions. You can run spark in interactive mode with immediate feedback, which is very handy while debugging. Hadoop MapReduce is programmed in Java. There are many tools available to ease the programmer like Pig and Hive. Pig provides a high level language for using Hadoop MapReduce operations and Hive is a distributed storage solution using SQL [2, 3].

Implementation

The most important thing for implementation is to reduce the amount of shuffling because this requires a lot network traffic which is relatively slow. In task one the total view and the most visited page per language is computed in one reduce by key operation. In task three a map-side join is performed to avoid a Join operation which requires shuffling. (<Lang>, <TotalRetweetCont>) had to be joined with (<IDofTweet>,(<Lang>,<Langcode>, <MaxRetweetCount>, <MinRetweetCount>, <Text>)) with <Lang> as key. In this case the first dataset is very small because the number of languages is limited. This dataset is broadcasted to all nodes such that a map-side join can be performed without shuffling.

The RDD's which are used more than once are persisted, to reuse the results for better performance.

Results

The results for the sample input of exercise one and two are the same as the provided sample output. The results or task two was evaluated manually. The code was run on a dual core computer with 4 GB of RAM.

Task one was tested on almost 6 million input records. It needed 11 seconds to finish. Task two was tested on the sample output of task 3. It took 6 seconds to finish.

Conclusion

Spark is the current state-of-art general engine for big data processing. Spark performs a lot better than Hadoop MapReduce especially for iterative patterns. I learned a lot about Spark because of the hands-on approach of the assignment.

References

1. https://www.xplenty.com/blog/2014/11/apache-spark-vs-hadoop-mapreduce/

2. https://hive.apache.org/

3. https://pig.apache.org/

4. http://blog.cloudera.com/blog/2015/03/how-to-tune-your-apache-spark-jobs-part-1/

5. http://spark.apache.org/docs/latest/programming-guide.html

6. https://spark.apache.org/docs/0.8.1/api/core/org/apache/spark/rdd/RDD.html

7. https://spark.apache.org/docs/0.7.2/api/streaming/spark/streaming/DStream.html

8. https://www.linkedin.com/pulse/batch-vs-stream-processing-luis-claudio-rodrigues-da-silveira