**SPARK**

**Project Work**

**Summary**

The aim of the project is to extract the top 20 topics that were discussed by the ex-senators using Spark. Since there is huge number files containing tons of text data related to the speech given by each ex-senators, computing it would take lot of time and storage. Hence, we have used **Spark** framework to do this task so that we do it quickly and efficiently. To Host our project we would require one master node and few worker nodes to split and do the tasks. So, we have used Amazon Elastic Compute Cloud (**Amazon EC2**), which is a web service that provides resizable computing capacity, i.e. servers in Amazon's data centers that we use to build and host our project. And for storing our ex-senators’ speech files we have used Amazon Simple Storage Service (**Amazon S3**).

**Development**

1. **Preparing the data**

We have got the data that we need to work on from the source below.

https://github.com/lintool/GrimmerSenatePressReleases/tree/master/raw

And uploaded the data to Amazon S3 at “s3://senatorbucket/input”

1. **Coding**

The code “SparkAction.scala” to get top 20 topics from the data is rewritten in scala language using Maven project. The algorithm behind it is given below.

* 1. Create SparkContext
  2. Create pressData RDD with key-value pairs of input text file path and its contents using Transformation “wholeTextFiles”.
  3. Create mappedData with senator’s name (derived from file path) and their file contents using Transformation “map”.
  4. Reduce the mappedData by key i.e combine the text file content per senator to authorGroup using Action “reduceByKey”.
  5. Remove stop words (i.e. words like is, are, was, the…) from authorGroup using Transformation “mapValues” to toPrint.
  6. Do word count on to Print using Transformation “mapValues” to finalData
  7. Take top 20 words with max count using Action “take” to “sortedMap”
  8. Save the results to "s3://senatorbucket/output/"

**Note**  
RDD is immutable. Hence, every step in scala code creates a new RDD.  
The program does two iterations of Transformations and Actions which is difficult to code in MapReduce.

1. **Preparing the jar file**

The Source code is compiled and jar file is generated under target folder.

The resulted jar file is uploaded to "s3://senatorbucket/jar/" so that it could be used to deploy it on Amazon cloud.

1. **Deployment**

A new cluster is created in Amazon EMP with 1 Master and 4 worker nodes with Spark installed on each system.

Deploy in this cluster by “Add Step” pointing to the jar file saved in S3 and the “SparkAction” class created in scala.

1. **Results**After the run completion the results will be saved in "s3://senatorbucket/output/"

**Evaluation**

The project took around 4 minutes on cloud cluster with 1 master and 4 worker nodes and on cloud cluster with 1 master and 2 worker nodes it took around 5 minutes. Whereas, on a single node cluster it took around 40 minutes.

With this we could see a significant performance improvement with the increase in the number of nodes.

**study Experience and analysis**

Initially, we have started this project by exploring the Hadoop MapReduce framework but we came across an article which mentions about Spark, which is an emerging technology that does the same work as Hadoop MapReduce, but more efficiently and it is way much easier to code due to its richness in APIs. More we dig in, more we are inspired with this new technology. Hence, we adopted our project work to apache spark.

Almost every tutorial about spark has an example of using either word count or inverted index programs. In order to realize the actual use of Spark we need to iterate the Transformation (i.e. map()) and Action (i.e. reduce()) at least twice. So with the senators’ data we added a layer of complexity by removing the stop words from the data.

After deciding on the main topic and problem statement, we need to choose the language to implement. Spark supports Java, Scala and Python. Our obvious choice was initially Java. But we begin to code, we found out that few tasks like doing the word count and removing the stop words can be easily done using Scala rather than java. One best feature of Scala which we found very interesting is the query operators (which are analogues to LINQ in C#) which manipulates with the dataset without having to iterate over. It results in much smaller and cleaner code. Hence, we ended up implementing the code in Scala.

Once we have our code finished and tested locally which is a single node cluster, we were looking for options to deploy in a cluster with more than 1 node. Since we have limited hardware and storage constraints, we ruled out having a cluster of physical devices with spark and also ruled out having a cluster of virtual machines with spark. Besides, setting up multiple machines in a short period of time is a tedious task.

Our next option is the cloud service. Two of the popular choices are Microsoft Azure HDInsight and Amazon web service EMR. On exploring both the options we found that Microsoft Azure HDInsight offers Hadoop MapReduce but Spark needs to be installed on the nodes manually. Whereas, in Amazon web service we can specify the cluster to install spark in the nodes automatically and it will be taken care of. Hence, we chose the Amazon’s trial service for deploying our project.

This project experience has given us an immense exposure to many latest technologies and services that are offered in the IT space. We had an opportunity to learn a new language scala, a new emerging technology spark and about amazon’s cloud service which is very simple and easy to use and how it helped us to overcome the hardware shortage very quickly. The Amazon’s EMR helped us to deploy our project effortlessly without having the hassle of setting up physical machine and configuring them.

The effort we have put on with this project will significantly help us in our future work with similar tasks. Also, we would like to extend the knowledge gained and the project experience in the near future.