Network Performance Analysis Using Spark

1. Introduction

In a real-time network architecture comprising of various connected devices, it is very important to analyze and improve the performance of the network in a timely manner. One of the ways to get the network information in real time is Logging. Every device in the network logs every action, timestamp, devices it communicated with etc These logs are generally saved on the local machine and then sent to the server for processing and analysis. Either way, it is very important that the logs have complete information about a device and its activity.

Generally these logs are collected over a period of time and then processed for analysis. This approach has a couple of problems. First, if a device runs into an error just after the logs are processed, the error is not known till the next time the logs are processed. Secondly, processing such huge data takes a lot of time on a server. There are several other bottlenecks in a network handling huge amount of data. A possible solution to these problems is to use a cluster-computing framework, such as Apache Spark, which has a lot of capabilities in handling big data processing.

2. Project Goals and Scope

The important goals that are focused upon in this RFP are developing the Apache Spark Ecosystem in the network and using its API for streaming and machine learning efficiently to process the big data in the network. Apache Spark is an open-source distributed cluster-computing framework with built in capabilities for Streaming, SQL, machine learning and graph processing. Apache Spark has streaming capabilities which can collect data from the network in real-time and process it simultaneously. This RFP proposes using Apache Spark to use as a means to monitor the network and process data as it is generated in real time. This will help the architecture to tend to the bottlenecks in the network. Figure 1 shows the general spark ecosystem.

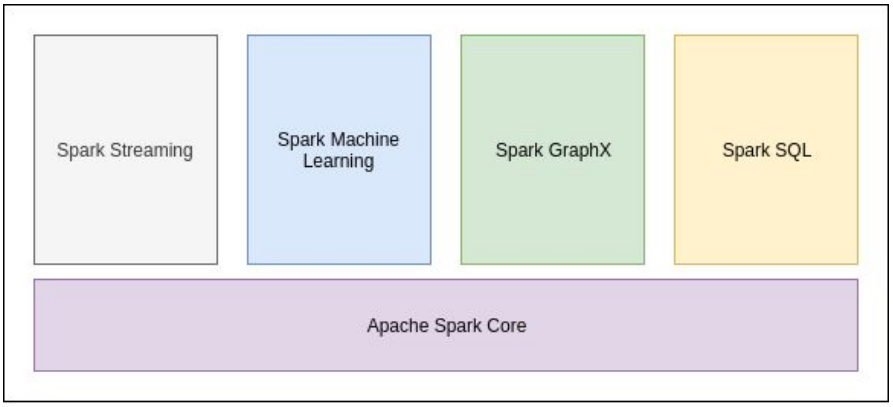

Figure 1

To monitor a network and analyze its performance, we use streaming and machine learning capabilities of Apache Spark. The streaming API of Spark helps us collect data from a network in real-time and process it simultaneously. The MLLib capability of Spark helps us run various machine learning algorithms on the processed data, derive and predict various details about the network.

2.1.1. Architecture of Apache Spark

Spark architecture consists of two main components known as Driver and Worker which act in Master-Slave fashion where “Driver” thread is the master and “Worker” thread is the slave. Driver gives tasks to worker to execute and worker threads execute these tasks on the big data provided. Figure 2 shows the architecture.

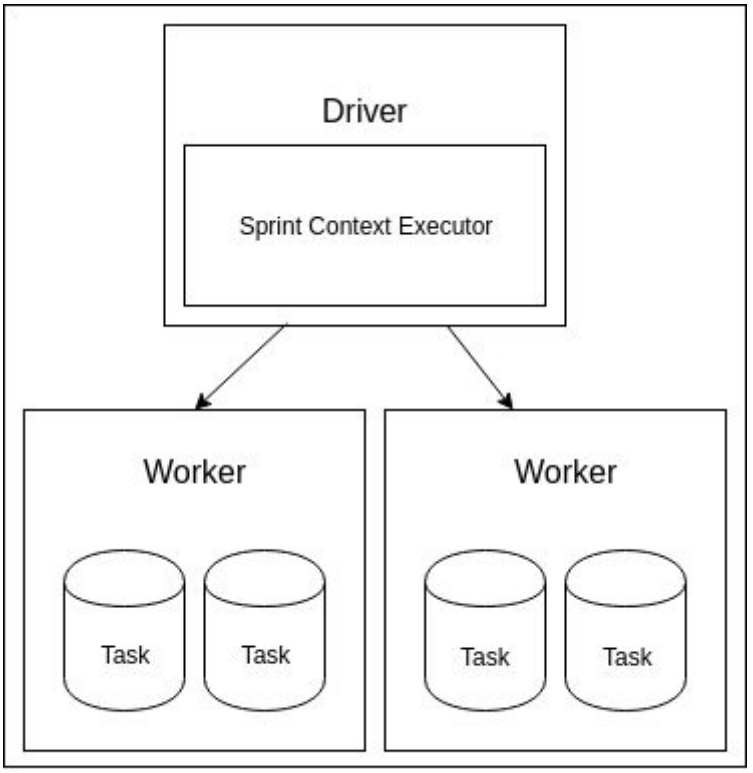

Figure 2

2.1.2. Lifecycle of Apache Spark instance:

The Apache Spark supports all the components required for this RFP to be optimal. There are few stages in the life cycle of Spark that are necessary to know about for deploying it in a server for processing and storage. Figure 3 shows the stages the data goes through in Apache Spark.

A. Data Sources: The data sources for this RFP can be the input network streams which carry the data from devices in the network. This helps in collecting data easily in real time providing fast processing of the data. The data sources are loaded on to the cluster and a Resilient Distributed Dataset (RDD) is created. A RDD is a collection of elements partitioned across the nodes of the cluster that can be operated on in parallel. RDDs can be transformed files from the Hadoop file system or an existing Scala collection in the driver program. These RDDs are immutable ad transformations are generated as Directed Acyclic Graphs. RDDs recover from failures and nodes are computed automatically in DAGs. The network logs streamed into the Spark framework are mostly a stream of characters or a text file which contain details according to format described during the building of the network. For example, a sample network log looks like,

1331901047.230000	CCHNF14C6RAO93DP7	192.168.202.76	68	192.168.202.1	67	00:26:9e:83:a2:30	192.168.202.76	0.000000	2767824/0
1331901117.740000	CouYOF1J4EnQkQNSl3	192.168.204.69	68	192.168.204.1	67	00:26:b9:da:95:2c	192.168.204.69	0.000000	2023309577
1331901120.620000	C9svD93TrEvPshF7Gf	192.168.202.102	68	192.168.202.1	67	f0:de:f1:2e:6a:5a	192.168.202.102	0.000000	7111068
1331901121.800000	C2nAD54rXz5nILppHh	192.168.202.76	68	192.168.202.1	67	00:26:9e:83:a2:30	192.168.202.76	0.000000	4022009768
1331901182.540000	CVRJN6491gIrhKWzHk	192.168.204.69	68	192.168.204.1	67	00:26:b9:da:95:2c	192.168.204.69	0.000000	3428947570

As seen in the sample data, it is comprised of columns displaying various details of a DHCP server of a network. It can help us identify various details about the network server.

B. Transformations: The RDDs provide transformations that are necessary for making the received data suitable for processing by converting them to datasets, dataframes etc. Some transformations that can be used are map, filter, textFile, flatMap etc. The RDDs are immutable which suggest that once enough data is collected and sent for processing, it cannot be changed.

C. Action: The actions, in this case, are algorithms to filter out the data from RDD and analyze various metrics to monitor the network performance. Each stream of input data that is received by spark can be processed using an algorithm to get the desired data/output from the input.

D. Output: The output of the spark algorithm can vary over a wide spectrum which is a very good response. It can vary from showing the time difference between request and response in a device to which device in network is suffering from bottlenecks due to hardware or software to checking which devices or ports in the network are failed based on heartbeat pings in the network. This output can be displayed to the user as list of timestamps, physical addresses of the devices in the network, ip address of the devices in the network etc. This output data that has been thoroughly processed in Spark can also be saved on another disk as processed data which can be used in future for processing the similar data very fast and/or for machine learning purposes.

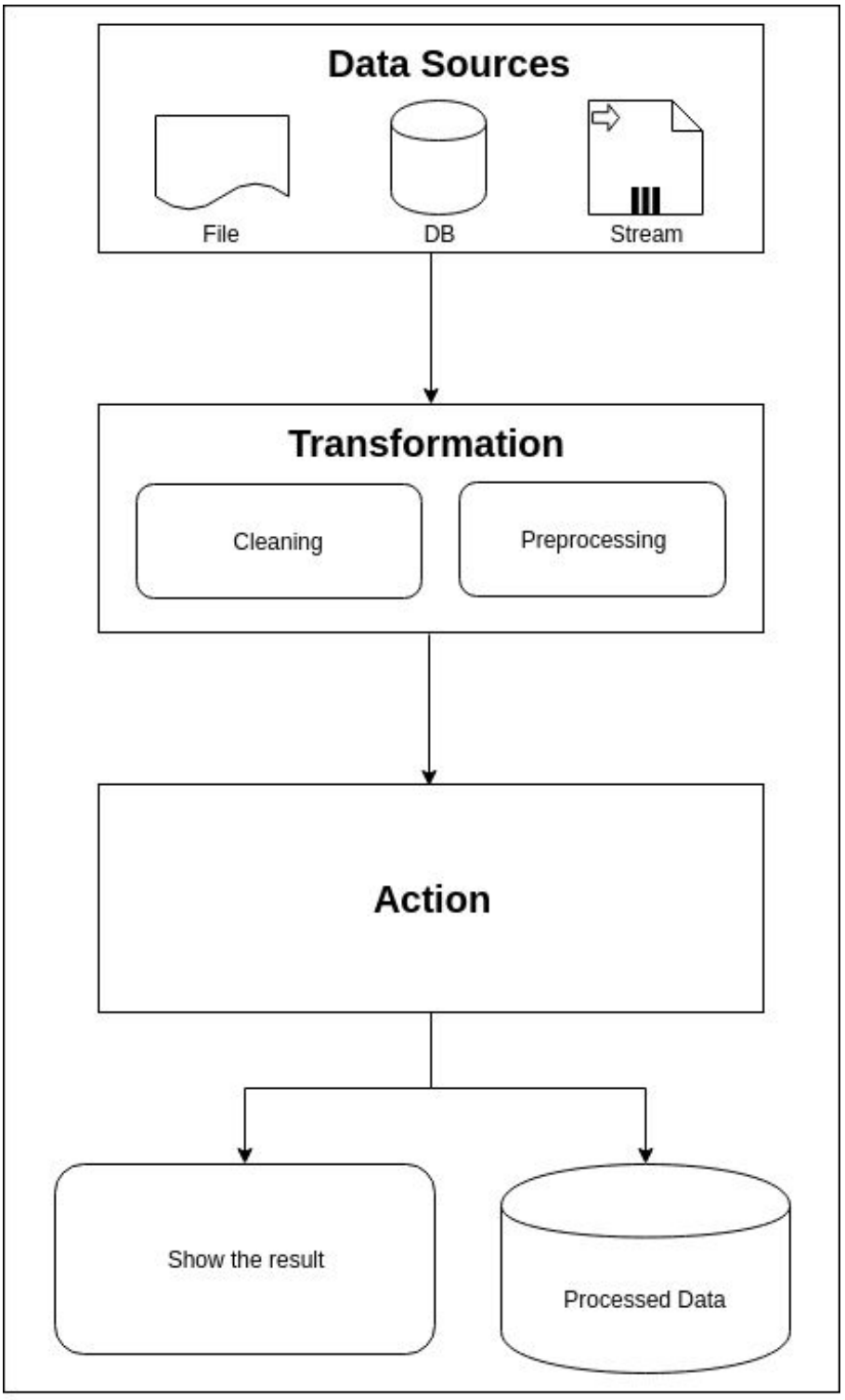
Data Sources
File
DB
Stream
Transformation
Cleaning
Preprocessing
Action
Show the result
Processed Data

Figure 3

3. Important Pros and Cons of this RFP:

Pros:

A. Efficient handling of the data generated in the network makes this proposal a preferable.

B. Use of all open source technology makes it cost efficient.

C. The model is highly extensible, scalable and reliable.

D. Multiple programming language support for languages such as Python, Java etc.

E. Support for graph data and relational database such as SQL.

F. Support for distributed data storage systems such as Hadoop File System.

G. Support for multiple input data formats such as text files, log files, stream of characters etc.

Cons:

A. It requires training and experience to handle Spark instance in a network.

B. Have to make sure that the machine running spark instance never shuts down and have multiple spark instances running on multiple machines.

C. Require machine with server grade hardware specifications

4. Scalability, Extensibility and Reliability

Since Spark supports receiving data from input stream from various devices, it becomes highly scalable. Adding new nodes to the network results in more input to the spark instance or server which can perform various algorithms on the data when coupled with distributed data storage system such as Hadoop, the whole architecture becomes very scalable for a large number of devices connected in the network. This idea of architecture can be extended by adding the machine learning capabilities using the MLLib library of spark support to machine learning. The processed data produced in the spark lifecycle is stored and various details can be predicted such as which sector of network can be under stress during a certain function being carried out or finding out which network area may need maintenance after a certain period of time.

5. Conclusion

In conclusion, Apache spark coupled with Hadoop for really large data, can be very beneficial and cost effective for an organization with huge data clusters that require processing and analysis. These details about the network can help the organization find out performance of the network under huge stress and how it can be improved and handled very efficiently. This type of analysis can be helpful in designing a very efficient network hierarchy and maintenance algorithm in the data cluster.