Assignment - 02 - Big Data

| Roll No. | 58 |
| --- | --- |
| Name | Ninad Rao |
| Class | D15-A |
| Subject | DS using Python Lab |
| LO Mapped | LO5: Design and Build an application that performs exploratory data analysis using Apache Spark  LO6: Design and develop a data science application that can have data acquisition, processing, visualization and statistical analysis methods with supported machine learning technique to solve the real-world problem |
|  |  |

**Date of Submission**: 10th April 2022

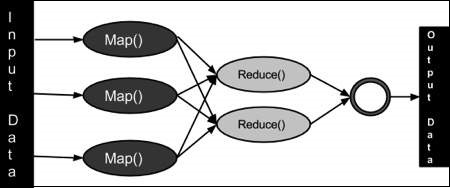
**Explain the MapReduce**

MapReduce is a processing technique and a program model for distributed computing based on java. The MapReduce algorithm contains two important tasks, namely Map and Reduce. Map takes a set of data and converts it into another set of data, where individual elements are broken down into tuples (key/value pairs). Secondly, reduce the task, which takes the output from a map as an input and combines those data tuples into a smaller set of tuples. As the sequence of the name MapReduce implies, the reduce task is always performed after the map job.

The major advantage of MapReduce is that it is easy to scale data processing over multiple computing nodes. Under the MapReduce model, the data processing primitives are called mappers and reducers. Decomposing a data processing application into mappers and reducers is sometimes nontrivial. But, once we write an application in the MapReduce form, scaling the application to run over hundreds, thousands, or even tens of thousands of machines in a cluster is merely a configuration change. This simple scalability is what has attracted many programmers to use the MapReduce model.

**Algorithm Process**

Generally the MapReduce paradigm is based on sending the computer to where the data resides. MapReduce program executes in three stages, namely map stage, shuffle stage, and reduce stage.



1. **Map stage** − The map or mapper’s job is to process the input data. Generally the input data is in the form of a file or directory and is stored in the Hadoop file system (HDFS). The input file is passed to the mapper function line by line. The mapper processes the data and creates several small chunks of data.
2. **Reduce stage** − This stage is the combination of the Shuffle stage and the Reduce stage. The Reducer’s job is to process the data that comes from the mapper. After processing, it produces a new set of output, which will be stored in the HDFS.

During a MapReduce job, Hadoop sends the Map and Reduce tasks to the appropriate servers in the cluster. The framework manages all the details of data-passing such as issuing tasks, verifying task completion, and copying data around the cluster between the nodes.

Most of the computing takes place on nodes with data on local disks that reduces the network traffic. After completion of the given tasks, the cluster collects and reduces the data to form an appropriate result, and sends it back to the Hadoop server.

**Compare between Map Reduce and Spark**

| **MapReduce** | **Spark** |
| --- | --- |
| It is a framework that is open-source which is used for writing data into the Hadoop Distributed File System. | It is an open-source framework used for faster data processing. |
| It is having a very slow speed as compared to Apache Spark. | It is much faster than MapReduce. |
| It is unable to handle real-time processing. | It can deal with real-time processing. |
| It is difficult to program as you require code for every process. | It is easy to program. |
| It supports more security projects. | Its security is not as good as MapReduce and continuously working on its security issues. |
| For performing the task, It is unable to cache in memory. | It can cache the memory data for processing its task. |
| Its scalability is good as you can add up to n different nodes. | It has low scalability as compared to MapReduce. |
| It actually needs other queries to perform the task. | It has Spark SQL as its very own query language. |

**Design a case study explaining the analysis performed on streaming data**

A **Case Study of Spark**: Understanding the Analytics Engine for Big Data and ML

Apache Spark is an open-source, distributed processing system and unified computing engine used for big data tasks. It utilizes in-memory caching and optimized query implementation for urgent queries for data of all sizes. In simple terms, Spark is a high-speed and general computing engine for large-scale data processing.

The high-speed part implies that it’s faster than traditional methodologies that work with big data, such as the traditional MapReduce. The secret to high-speed computational power is that Spark runs on Random Access Memory (RAM). This makes processing much faster than on disk drives.

The general part implies that it can be used for accomplishing different tasks such as running distributed SQL, employing machine learning (ML) algorithms, building data pipelines, working with graphs or data streams, ingesting data into a database, and much more.

Three key components make Spark the best in solving big data problems at scale, which encourage many businesses working with huge volumes of unstructured data to include Apache Spark into their technology stack.

1. **Spark is a unified, distributed computing engine for working with Big Data**

Spark is designed to support many data analytics tasks, including simple data loading, distributed SQL queries, machine learning algorithms, and streaming computation on the same computing engine and with a specific set of APIs.

The main motive of this goal is that real-world data analytics tasks whether they are intuitive analytics in a tool like Jupyter Notebook or conventional software development for production applications seek to combine various processing types and libraries. Spark’s distributed nature makes these tasks more efficient and simpler to solve. For instance, if you use a SQL query for data loading and then evaluate a machine learning model over it through Spark’s library, Spark combines these steps into one scan over the data.

Moreover, data scientists can leverage a wide range of libraries when doing modeling. Additionally, web developers can benefit from unified frameworks such as Django or Node.js.

2. **Spark’s libraries offer a huge array of functionalities**

At present, Spark’s standard libraries include the ones in the open-source project. Since its inception, Spark’s core engine has gone through a myriad of changes, but the libraries have grown to include more features and offer more types of functionalities, thereby turning it into a multifunctional data analytics tool.

Spark includes libraries such as Spark SQL (structured data), GraphX (graph analytics), MLlib (machine learning), and Spark Streaming and the newly Structured Streaming for stream processing. Apart from these, there are several other open-source external libraries ranging from machine learning algorithms to connectors for different storage systems.

3. **Spark streamlines its core engine for computational efficiency**

This means that Spark only deals with data loading from storage systems and performs computational tasks on it, not permanent storage as the end itself. The engine is compatible with a lot of persistent storage systems, including cloud storage systems such as Amazon S3 and Azure Storage, Apache Cassandra, distributed systems such as Apache Hadoop, and message buses such as Apache Kafka.

In any case, Spark neither stores data over the long haul nor favors one of these. The key objective here is that the majority of data already resides in a combination of storage systems. It is expensive to move data, so Spark emphasizes performing computations over the data, irrespective of where it is stored.

Since Spark emphasizes computational tasks, it differs from traditional big data software platforms such as Apache Hadoop. It includes both a computing system (MapReduce) and a storage system (the Hadoop file system). The Hadoop file system is designed for inexpensive storage over clusters of Defining Spark 4 commodity servers, which is closely integrated with MapReduce. Because of this setting, it is difficult to run one of the systems without the other or to write applications that access data stored anywhere else.

**Findings and Result**

Spark is a highly adaptable data analysis engine for data processing. It is popular among companies and businesses because it has two major advantages. The first advantage is speed. Spark’s in-memory data engine allows it to reduce processing time when analyzing large data sets. Its second advantage is its developer-friendly API. This engine can be integrated with many different data analysis languages such as Python, R, and JavaScript.

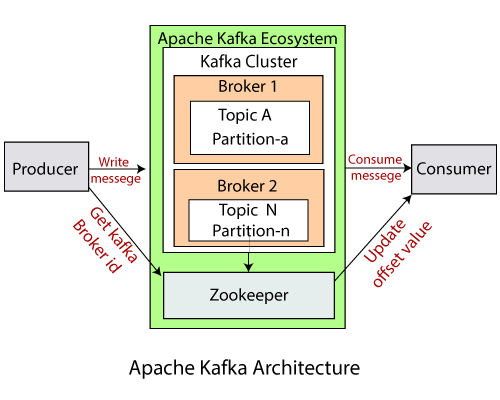
The most popular use case for Spark is processing streaming data. Being able to stream and analyze data in real-time has become critical for companies that need to process huge volumes of data on a daily basis, and Spark’s streaming capabilities help handle this additional workload. Tech giants such as Netflix, Yahoo, and eBay already use Spark on a large scale. The general ways enterprises use Spark Streaming today are Streaming ETL, Data Enrichment, Trigger Event detection, and Complex Session Analysis.

With the vast expansion of Big Data and the unprecedented speed of computational power, software tools such as Apache Spark and all other Big Data Analytics engines will soon be a necessity to data scientists. Moreover, it will quickly become the industry standard for performing Big Data Analytics and solving robust and complex business problems at scale in real-time.

**Explain the architecture** - **Apache Kafka**

Apache Kafka is a software platform which is based on a distributed streaming process. It is a publish-subscribe messaging system which lets exchanging of data between applications, servers, and processors as well. Apache Kafka was originally developed by LinkedIn, and later it was donated to the Apache Software Foundation. Currently, it is maintained by Confluent under Apache Software Foundation. Apache Kafka has resolved the lethargic trouble of data communication between a sender and a receiver.

As different applications design the architecture of Kafka accordingly, there are the following essential parts required to design Apache Kafka architecture.

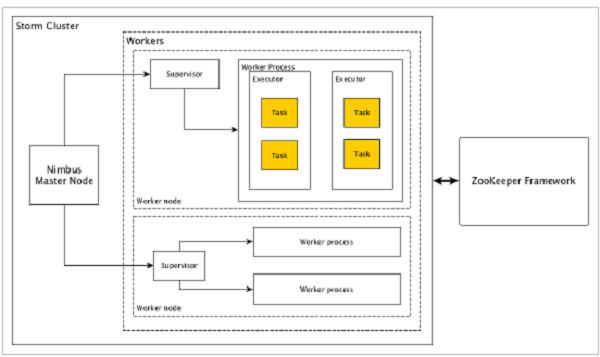


1. **Data Ecosystem** - Several applications that use Apache Kafka form an ecosystem. This ecosystem is built for data processing. It takes inputs in the form of applications that create data, and outputs are defined in the form of metrics, reports, etc.
2. **Kafka Cluster** - A Kafka cluster is a system that comprises different brokers, topics, and their respective partitions. Data is written to the topic within the cluster and read by the cluster itself.
3. **Producers** - A producer sends or writes data/messages to the topic within the cluster. In order to store a huge amount of data, different producers within an application send data to the Kafka cluster.
4. **Consumers** - A consumer is the one that reads or consumes messages from the Kafka cluster. There can be several consumers consuming different types of data from the cluster. The beauty of Kafka is that each consumer knows where it needs to consume the data.
5. **Brokers** - A Kafka server is known as a broker. A broker is a bridge between producers and consumers. If a producer wishes to write data to the cluster, it is sent to the Kafka server. All brokers lie within a Kafka cluster itself. Also, there can be multiple brokers.
6. **Topics** - It is a common name or a heading given to represent a similar type of data. In Apache Kafka, there can be multiple topics in a cluster. Each topic specifies different types of messages.
7. **Partitions** - The data or message is divided into small subparts, known as partitions. Each partition carries data within it having an offset value. The data is always written in a sequential manner. We can have an infinite number of partitions with infinite offset values. However, it is not guaranteed that to which partition the message will be written.
8. **ZooKeeper** - A ZooKeeper is used to store information about the Kafka cluster and details of the consumer clients. It manages brokers by maintaining a list of them. Also, a ZooKeeper is responsible for choosing a leader for the partitions. If any changes like a broker die, new topics, etc the ZooKeeper sends notifications to Apache Kafka.

**Explain the architecture Apache Storm**

Apache Storm is a distributed real-time big data-processing system. Storm is designed to process vast amounts of data in a fault-tolerant and horizontal scalable method. It is a streaming data framework that has the capability of highest ingestion rates. Though Storm is stateless, it manages distributed environment and cluster state via Apache ZooKeeper. It is simple and you can execute all kinds of manipulations on real-time data in parallel.

One of the main highlights of the Apache Storm is that it is a fault-tolerant, fast with no “Single Point of Failure” (SPOF) distributed application. Let’s have a look at how the Apache Storm cluster is designed and its internal architecture. The following diagram depicts the cluster design.



Apache Storm has two types of nodes, Nimbus (master node) and Supervisor (worker node). Nimbus is the central component of Apache Storm. The main job of Nimbus is to run the Storm topology. Nimbus analyzes the topology and gathers the task to be executed. Then, it will distribute the task to an available supervisor.

1. **Nimbus** - Nimbus is a master node of the Storm cluster. All other nodes in the cluster are called worker nodes. Master node is responsible for distributing data among all the worker nodes, assigning tasks to worker nodes and monitoring failures.
2. **Supervisor** - The nodes that follow instructions given by the nimbus are called Supervisors. A supervisor has multiple worker processes and it governs worker processes to complete the tasks assigned by the nimbus.
3. **Worker process** - A worker process will execute tasks related to a specific topology. A worker process will not run a task by itself, instead it creates executors and asks them to perform a particular task. A worker process will have multiple executors.
4. **Executor** - An executor is nothing but a single thread spawn by a worker process. An executor runs one or more tasks but only for a specific spout or bolt.
5. **Task** - A task performs actual data processing. So, it is either a spout or a bolt.
6. **ZooKeeper framework** - Apache ZooKeeper is a service used by a cluster (group of nodes) to coordinate between themselves and maintain shared data with robust synchronization techniques. Nimbus is stateless, so it depends on ZooKeeper to monitor the working node status. ZooKeeper helps the supervisor to interact with the nimbus. It is responsible to maintain the state of nimbus and supervisor.