FRAMEWORK COMPARISON FOR DATA STREAMING & IMPLEMENTATION USING APACHE SPARK IN CLOUD (Cloud: Amazon AWS)

Project: Uber Data Analysis

Hochschule Schmalkalden

Schmalkalden, Thüringen

MSc Informatik WiSe 2019/20

Prof. Dr.-Ing. Dr.phil. Michael Cebulla

Team Members:

Madhan Raj Moorthy

Table of Contents:

1. Abstract

Basically, data are the units of information, which is to transfer, display, to the end users. Big Data deals with large-volume datasets, complex and growing data with multiple, independent sources. As per the increasing trend in data requirements for several purposes, the data is brought up in huge volume (Big Data). When it comes to handle/process more volume of data, the need of respective efficient tools, frameworks, cloud, databases are needed. This paper will discover about the elements of the popular big data processing frameworks.

Keywords
Big Data, Simulated Data (Uber Data), Data Streaming, Apache Spark, Apache Storm, Apache Flink, Apache Hadoop & Cloud.

2. Introduction

Over the past decade, there is a lot of data, all the time, growing at 50 percent a year, or more than doubling every two years, estimates International Data Corporation, a technology research firm. It's not just more streams of data, but entirely new ones such as Internet of Things (IoT), Industrial Internet, Ambient Assisted Living (AAL), e-health and telemedicine. Huge data can be generated in many mediums as above, companies(enterprises), applications, software's. For example, there are numerous digital sensors globally used in automobiles industries, electrical meters and shipping crates. Where these sensors can measure the functionalities like location, movement, vibration, temperature, humidity.

Data plays an enormous role in several fields and its necessities are increasing reckless and the volume of data generated is huge amount, so processing/streaming of the data becomes challenging.
The process of real-time data streaming is to handle and process huge volumes of data quickly. There is an increasing need for taking out information from functional data in real time, which is crucial in fast situations. The faster one can bind insights from data, the larger benefit in driving value, taking, reducing costs, and increasing efficiency.

Nowadays, the data streaming with huge volume of data remains an open difficulty with more efficiency and scalability(storage) in mind. Many companies, platforms have resorted to cloud computing to satisfy efficiency and scalability requirements. Cloud computing is a model which enables ubiquitous, suitable, on-demand network access to a shared pool of configurable computing resources for example: storage, networks, servers, applications and services. The real time processing of data requires tools

and frameworks by which we can get efficiency and scalability. There are several technologies, tools and frameworks like Apache Storm, Apache Flink, Apache Spark, Apache Kafka that provides users, the ability to access large volumes of data and visualizing results in real time. This paper presents about the frameworks and an implementation with one framework Apache Spark.

3. Components Covered:

3.1.Big Data:

In general, the terminology known as Big Data refers to a large amount of data sets that comprises of mixed formats: structured, unstructured and semi-structured data. Big Data is complex in nature and both powerful technologies and advanced algorithms are required. Data can be changed as per the source like organizational data, data from online streaming websites, social medias and so on. Big data is all about everything that does not work with traditional technology because of the size of the data, such as streaming, collecting, storing, searching, distributing, analyzing, and visualizing large amounts of data.

The development of big data shows the sharp increase in global data volumes. Most sources are responsible for this: Networks, social media, sensor data, machine data, log data, World Wide Web (WWW) and so on. The turnout of huge data and its requirement of processing are multiplying enormously. Standard databases and tools are struggling to cope with the rising tide of data. The big data's 6V's plays the significant role to explicit the huge data and its processing.

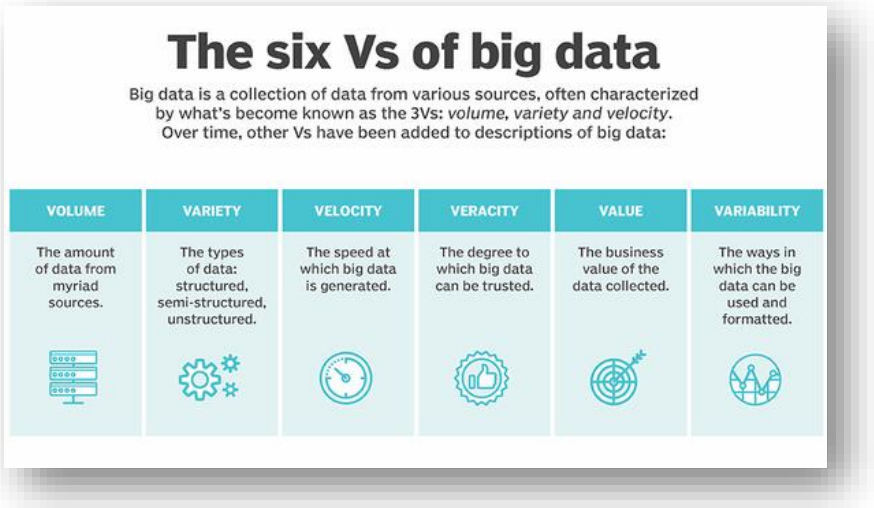

Fig 1: 6V's of Big Data

The above displays the 6 V's of big data with its tasks. Recently, the capability to collect and attain data at an unusual rate in almost all aspects of our life such as social interactions, financial transactions, environmental monitoring, etc. Big Data is globally used to join the data which is being analyzed together and to generate sureness for improved estimates by using frameworks and tools. When the huge data comes into the systems, processing, is the most important task that should be completed to handle those bug volumes of data. Primarily, when it comes to real time/online streaming for big data, several frameworks and tools exist to handle and process those data.

Volume:

- Ability to ingest, process and store large amount of dataset.
- Machines, networks, human interactions on system generates data.
- Data can be measured in terms of petabytes or Exabyte.

Variety:

- Various source and types of data are referred which may be either structured or unstructured.
- Unstructured data generates problems for storage, data mining and analyzing the data.
- The type of data is growing fast with enormous growth of data

Velocity:

- Defines the data generation speed and frequency of delivery.
- The data flow is very large and continuous which is valuable to researchers as well as business, for making decision, strategic competitive advantages and ROI.
- Data processing with high velocity tools known as Streaming analytics were introduced.

Veracity:

- States the biases, noises and irregularity in data.
- This is where we are required to be able to identify the importance of data and ensure data cleansing is done to only stored valuable data.
- Verification of the data whether it is suitable for its intended purpose and usable within the analytic model.
- The data should be tested against a set of distinct criteria.

Variability:

- Refers to establishing whether the contextualizing structure of the data stream is regular and dependable even in conditions of extreme unpredictability.
- Defines the importance to obtain meaningful data considering all possible circumstances.

Value:

- Refers to purpose, scenario or business outcome that the analytical solution must address.
- In order to meet the ethical considerations analysis needs to be performed

3.2.Data Streaming:

A stream can be defined as a continuous inward flow of unbounded data. Large amount of data created by thousands of sensors, tools, dashboards and sending those data records can all be defined as streaming data. Overall, these data should be processed very efficiently.

3.2.1.Data Streaming feature:

Data streaming is a very potent tool, but several issues arise when working with streaming data sources. Below is a list of what to keep in mind and plan for when data streaming:

- Design your implementation with scalability in mind.

- Plan for data durability
- Plan and implement fault tolerance in the storage and processing layers

3.3.Real-Time Data Streaming:

Real-time data streaming involves big bulks of data that are processed quickly such that any company or institution extracting information from that data can respond to any varying situations in real time. In order to enable the organizations to notice and react to any fraudulent activity and potential threats, also to enhance the business benefits, large masses of data are stream processed. Real-time data streaming is implemented by incorporating continuous queries that work according to time and buffer windows. This whole process contrasts the traditional database design where data was first stored and indexed before being processed. Real-time data streaming processes the data while the data is moving through the server. Real-time data streaming has various applications. Some of them are Internet of Things, Network monitoring, Risk management, Fraud detection, E-commerce, pricing and analytics.

3.4.CLOUD COMPUTING

In reference to National Institute of Standards and Technology (NIST), Cloud computing is a model which enables ubiquitous, suitable, on-demand network access to a shared pool of configurable computing resources. For example: storage, networks, servers, applications and services. These can be quickly provisioned and deployed with less management effort or through service provider relations. The cloud model comprises five significant characteristics, three service models, and four deployment types.

The important characteristics are detailed in the following:

- On-demand self-service. This helps the customer have easy and ready access to tools and recources with minimal interaction with the service provider.
- Broad network access. The customer should be able to access the tools and resources through internal or public networks and also be able to access other required services using the desired networks.
- Resource pooling. Same resources are readily made available to customers to help with running all required services.
- Rapid elasticity. Customer server and services can easily be scaled up or down depending on the everchanging customer demands.
- Measured service. Systems and resources can easily be monitored and measured.

The Service Models include:

- Software as a Service (SaaS). These are mostly the provision of applications with specific functions for example, an email client.
- Platform as a Service (PaaS). This is the provision of web, application and database software to the customer.
- Infrastructure as a Service (IaaS). This includes provider managed storage, network, servers and others to help a customer deploy their data or application on.

Deployment Models:

- Private cloud. Infrastructure which is authorized for use of a single institution or a company.
- Community cloud. This is a cloud infrastructure which is provisioned only for use by a specified group of consumers from companies that have some shared interests.
- Public cloud. This is a cloud infrastructure that is provisioned for use by the general public.
- Hybrid cloud. This cloud infrastructure is usually a combination of at least two of the other cloud deployment models.

Some available cloud computing services in the world today are AWS Cloud Services, Microsoft Cloud Services, IBM Cloud, Amazon Elastic Compute Cloud, Citrix Cloud Platform, Joyent Cloud and Verizon Cloud and the Google Cloud Platform.

4. Frameworks

As we defined the meaning of Big Data in the previous stage, it is now important to illustrate its characteristics. It is composed of generic Big Data requirements.

4.1.APACHE HADOOP

HDFS is used to for store the big data which uses Hadoop technology. It is good and Map-reduce. As Hadoop can store and process many petabytes of information, where the processing speed is fast, and it takes only a few seconds to operate. Hadoop's main task is to process and calculate huge amount of data proficiently using clusters. Mainly it prevents any form of editing the information which is already stored in the HDFS system during the time of processing.

In the year 2008, Apache Hadoop was well-defined by Doug Cutting and Mike Cafarella as an open source framework. This usually collects and processes the distributed data through a unit of host machines in the hardware layer known as clusters or nodes. It provides a distribution services machine not only to one service. Hence it can work in parallel by using clusters or nodes.

Hadoop framework uses MapReduce as a search engine. Firstly, it was introduced as an algorithm for the parallel processing of considerable raw data volumes by Google. Later it became MapReduce which is well-known currently. This engine indulgence data in the form of entries and processes them in three different stages:

- **Map** (Known for its preprocessing and filtration of data).
- **Shuffle** (Includes the worker nodes sort data, each one agrees with one output key, which results from the map function).
- **Reduce** (the reduce function is set by the user and defines the result for separate groups of output data).
- Most of all values are returned by Reduce (functions are the final result of the MapReduce task**).**

Hadoop framework is good for reliable, scalable and for distributed calculations. Moreover, it can be developed as common-purpose file storage. Also, it can store and process petabytes of data. The result consists of three main components:

- HDFS file system which is responsible for the storage of data (Hadoop cluster).
- MapReduce system expected to process large volumes of data in cluster.
- YARN, a core that supervises resource management.

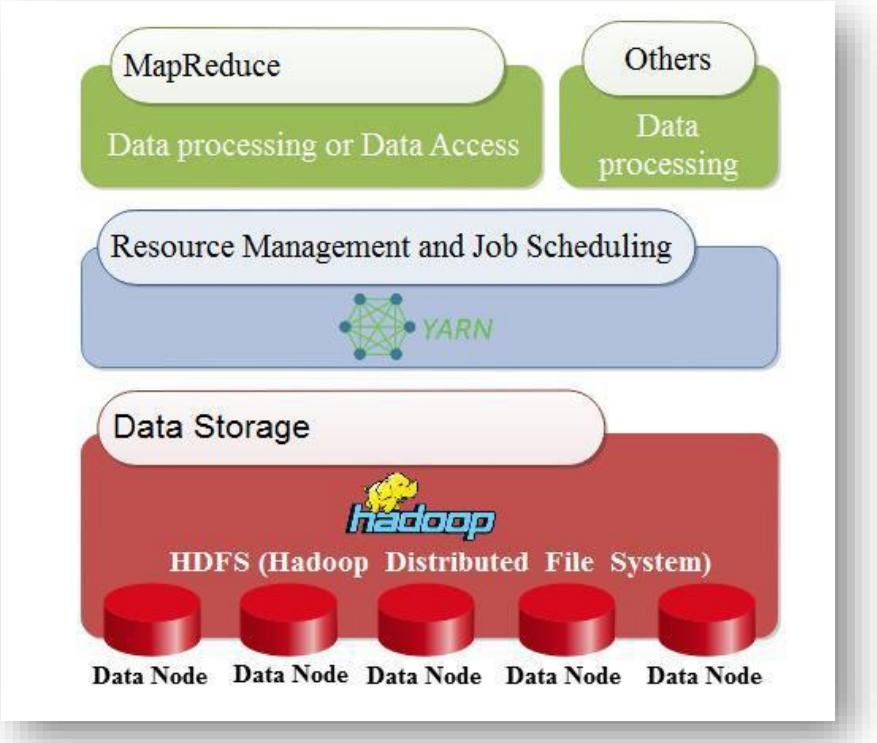

Fig 2: Hadoop Architecture

The above figure illustrates the three main layers of Hadoop framework. The first layer is the data storage layer used for collecting data that contains Hadoop Distributed File System (HDFS). The second one is the YARN infrastructure, that provides arithmetic resources for job scheduling such as CPU and memory. The third is MapReduce, which is used for processing data (software layer) with other processes.

Features:

- While using HTTP proxy server , the authentication improves
- Specification for Hadoop Compatible Filesystem effort
- Support for POSIX-style filesystem extended attributes
- It brings Flexibility in Data Processing
- It allows for faster data Processing

Pros: It include cost-effective solution, high throughput, high scalability, supports multi-language, compatibility with most developing technologies in Big data services, fault tolerance, Suitable for research & development.

Cons: It include vulnerability to security breaches, suffers processing overheads because it does not perform in-memory computation. It is not suited for stream processing and real-time

processing. Having issues in processing small files in large numbers. Less functionality and can't write data on its own, also solution is not customized.

4.2.APACHE HIVE

Apache Hive was formed by Facebook to merge the scalability of most popular Big Data frameworks. It is an engine which can make SQL-requests to cables of MapReduce tasks.

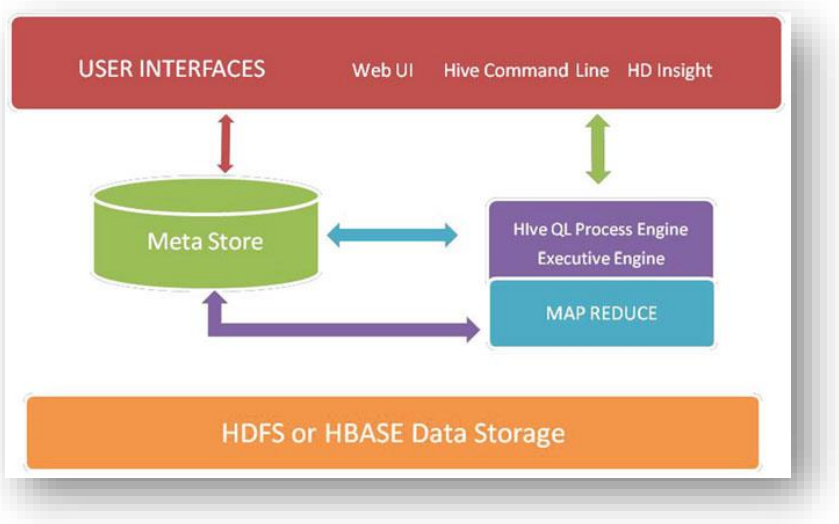

Fig 3: **Apache Hive Architecture**

Hive architecture, it has three portions in the architecture. In the meta portion, it stores information about the stored data. HiveQL is like other SQL it uses conversant RDB HiveQL is like SQL for querying on schema info on the metastore.

The engine includes such components as:

- Parser (that sorts the incoming SQL-requests)
- Optimizer (that enhances the incoming requests for high efficiency)
- Executor (that inaugurates the tasks in MapReduce)

Hive can be combined with server part of Hadoop for the analysis of huge data volumes.

Apache Hive enables to run queries and be able to handle huge datasets using meek commands same as SQL. We have mentioned the Hive part under Hadoop since, it works

with Apache Hadoop as a Data Warehouse. Apache Hive is a data warehouse interface for MapReduce programing as same as the traditional data warehouse methods.

Features:

- It cares SQL like query language for interaction and Data modeling
- It collects language with two main elements which are map, and reducer
- It permits defining these tasks using Java or Python
- Hive is intended for managing and querying only structured data

Pros include own query language HiveQL like SQL, suited for data-intensive jobs, support for a wide range of storages, shorter learning curve

Cons: It is not suited for most of the online transaction processing.

4.3.APACHE SAMZA

Apache Samza was first developed at LinkedIn, and then it was donated to the Apache Software Foundation in 2013. It became an important Apache project in 2015. Samza is now used in production in many companies for example LinkedIn, Netflix, Uber, and TripAdvisor and so on. It is also used by eBay and TripAdvisor for fraud detection. A large portion of its code was employed by Kafka to build a competitive data processing framework called Kafka Streams.

Apache Samza is an open source framework for the distributed processing of huge bulks of event streams**.** It is mainly designed goal is to support high throughput for a wide range of processing patterns, concurrently providing operational robustness at the massive scale needed by Technological companies.

Apache Samza can also be described as a stateful stream processing Big Data framework that was developed in conjunction with Kafka. The Kafka portion is intended to provide data serving, buffering, and fault tolerance. The pair is designed to be used in scenarios in which rapid single-stage processing is required. Together with Kafka, Samza can be used with low latencies. Samza also saves local states when processing streams and this also provides some extra fault tolerance.

Samza was designed for Kappa architecture (a stream processing pipeline only) but is not limited only to it. Samza uses YARN to assign resources. A Hadoop cluster is required for it to work, and that means the features provided by YARN can be relied upon.

Application areas for Apache Samza:

Some of the application areas for Apache Samza are shown below:

- Complementing events with information from a database, for example, adding to user click events more information about who performed the action.
- Combining related events, for example, joining an event showing an email that was sent with any event that shows any user clicking links in that email.
- Evaluating how often an event occurs, for example, counting the number of times an item has been clicked or viewed.
- Using machine learning systems for event classification, for example, filtering or spam messages.

A Samza job contains of a set of Java Virtual Machine (JVM) instances. Each instance is called a task. These tasks each processes a part of the input data. The programming code in each instance consists of the Samza framework and user code that does the required application-specific functionality.

StreamTask, the Java interface that defines the method process() is the main API used for the user code.

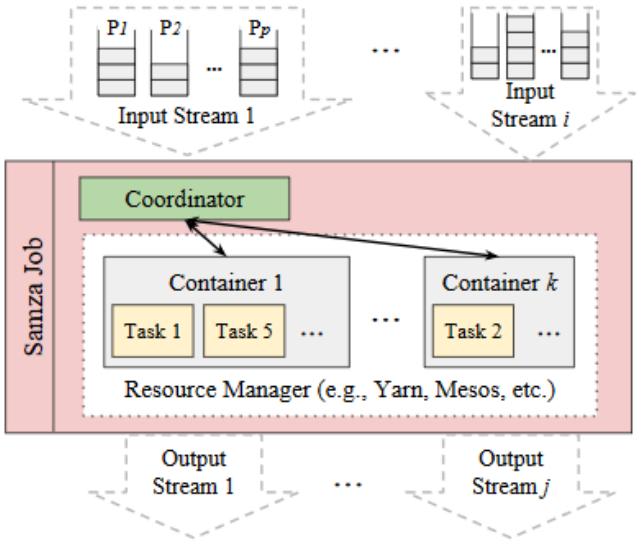

Fig 4: **Apache Samza Architecture**

When a Samza job has been deployed and initialized, process () method is called by the framework once for each message in any of the input streams. This method will result different outcomes, including the querying or updating of the local state and sending messages to output streams. This implementation is like a map task in the MapReduce programming model. The difference is that a Samza job has an input that is typically never ending (this is referred to as Unbounded)

```
class SplitWords implements StreamTask {

  static final SystemStream WORD_STREAM =
    new SystemStream("kafka", "words");

  public void process(
        IncomingMessageEnvelope in,
        MessageCollector out,
        TaskCoordinator _) {

    String str = (String) in.getMessage();

    for (String word : str.split(" ")) {
      out.send(
        new OutgoingMessageEnvelope(
          WORD_STREAM, word, 1));
    }
  }
}
```

```
class CountWords implements StreamTask,
                            InitableTask {

  private KeyValueStore<String, Integer> store;

  public void init(Config config,
                   TaskContext context) {
    store = (KeyValueStore<String, Integer>)
      context.getStore("word-counts");
  }

  public void process(
        IncomingMessageEnvelope in,
        MessageCollector out,
        TaskCoordinator _) {

    String word = (String) in.getKey();
    Integer inc = (Integer) in.getMessage();

    Integer count = store.get(word);
    if (count == null) count = 0;
    store.put(word, count + inc);
  }
}
```

Figure 5: Example of Running code: 2 Operators using Samza's StreamTask API

Similar to MapReduce, each Samza task is a single-threaded process that run through over a sequence of input data. The inputs to a Samza job are partitioned, and each input partition is processed by only one processing task.

The log interface concludes that each partition coming from the input is an ordered sequence of records and each record is related to a steadily increasing sequence number or identifier (which is known as an offset). Due to the fact that the records in each partition are sequentially processed, a job can easily measure progress by writing the offset of the last processed record to storage intermittently. When a stream processing task is then restarted, it will then continue consuming the input from the last recorded offset.

Typically, Samza is implemented in partnership with Apache Kafka. The main catch is that Samza's stream interface is pluggable. This means that besides Kafka, Samza can use any

storage or messaging system as input, only if that system complies with the partitioned log interface. Naturally, Samza reading of files from the Hadoop Distributed File system (HDFS)as input, similar to MapReduce jobs, with rivalling performance metrics is possible.

Processing the stream of writes into a database helps jobs to keep external indexes or views onto data in a database and is very applicable in combination with Samza's support for local state.

Even though every partition of an input stream is given to a single task of a Samza job, the resultant output partitions are not bound to tasks. This means, when a StreamTask produces output messages, they can be assigned to any partition of the output stream. This helps to group related data items into the same partition.

If the stream tasks are made into multistage processing channels, the output of one task is then the input to another task. Samza does not have its own message transport layer to help deliver messages between the numerous stream operators. Kafka is used to achieve this, since Kafka writes all messages to disk. This makes it provide a large buffer between stages of the processing channel, which is only limited by the disk capacity on the Kafka brokers. By default, Kafka has been designed to retain several days or weeks' worth of messages for any subject or topic. This makes sure that if one stage of a processing channel fails or is running slow, Kafka can easily buffer the input to that stage while leaving enough time for possible problem resolution.

Messages are only discarded if the failed or slow processing stage is not resolved within the required reservation period of the Kafka topic. With this occurrence, dropping messages is advisable because it keeps the fault isolated. The other solution would be to keep messages indefinitely until the fault is repaired. This would lead compromise existing resources (running out of memory or disk space) and can cause increased failure in the processing structure and other parts of the system.
Thus, Samza's use Kafka's on-disk logs to transport messages is an important part in its scalability.

Features:

The main features of Samza are:

- Adequate support for many application states, as most applications need to store or access different states along with their processing.
- Fast times to recover from failures and to restart jobs.
- Scalability. Samza can handle large data volumes and large input sources.

Pros: The benefits of Samza include the ease of operation, high performance, and horizontal scalability. The ability to run the same code for batch processing and streaming data is a plus. Its pluggable architecture gives it an advantage over the other frameworks.

Cons: Samza suffers in extreme cases of low latency processing.

4.4.APACHE STORM

Apache Storm is an open source framework that was intended for processing streaming data in real-time written in Clojure language. Nathan Marz devised an idea to develop a stream processing system in December 2010, that can be accessible as a single program.

Apache Storm allows developers to build real-time distributed processing systems, which can process the unbounded streams of data with less computation time. It is also known as Hadoop for real-time data.

Apache Storm is easy to use, highly scalable and provides low latency with assured data processing. It provides a simple architecture to build applications called Topologies. It allows developer to develop their logic virtually in multiple programming language, which supports communication over a JSON-based protocol over stdin & out. Apache Storm was officially included in Apache catalogue of frameworks

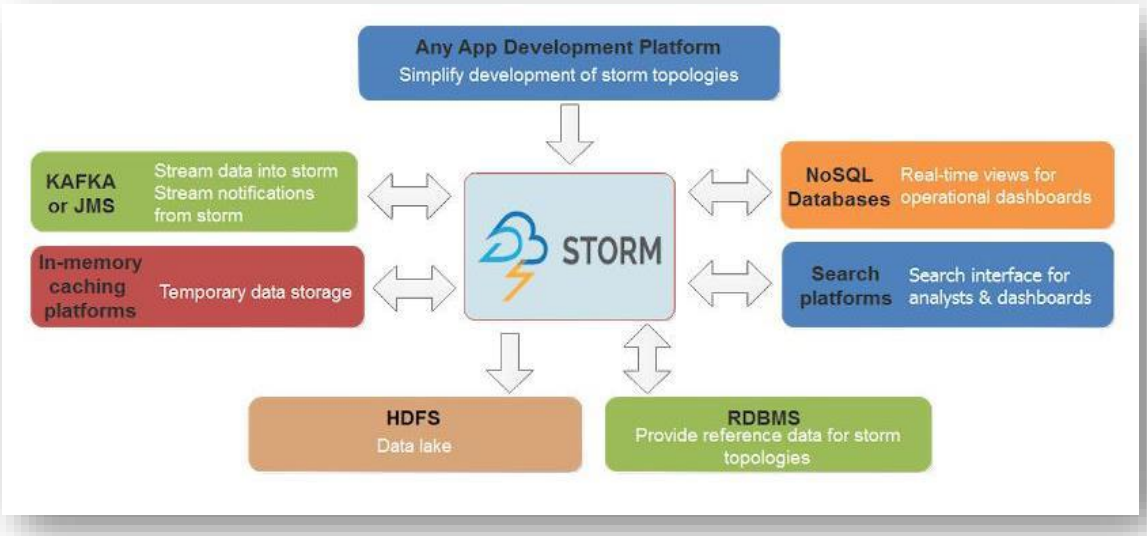

Fig 6: **Apache Storm Architecture**

Storm can work with any programming language and on multiple application development platforms. Data loss will be less.

Below are some key attributes, which make Apache Storm as a significant tool for processing real-time unbounded data:

- Fast – known for processing millions of byte data per second
- Fault-tolerant – holds the track of all worker nodes. Apache Storm restart the process on another node, whenever an active node die.

Below are some main criteria, decides the usage of Apache Storm:

- High fault tolerance
- Processing Model: Real-time stream processing model
- Language dependency: Applicable to any programming language
- Reliable: At least once, each tuple of data must be processed.
- Scalability: High scalability.

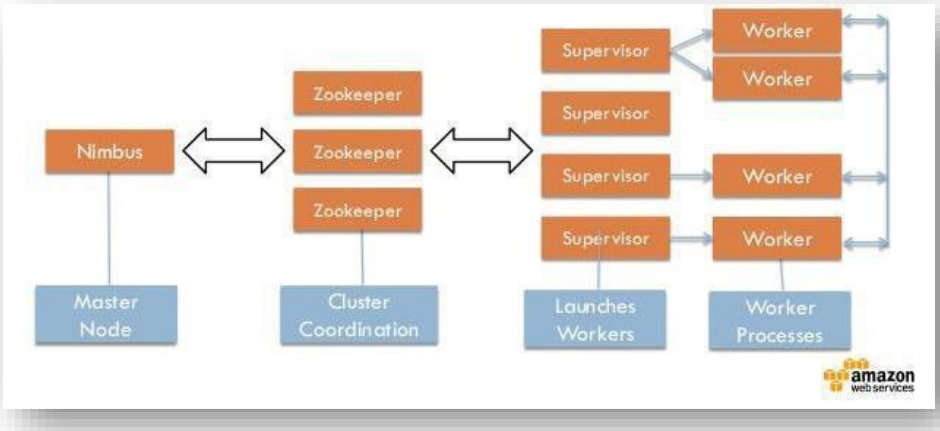

Fig 7: **Apache Storm Architecture**

The above diagram demonstrates two types of nodes: The master node and worker node respectively. The master node is used for monitoring failures, taking the responsibility of distributed node, and specifying each task for individual machine. All these tasks are collectively called Nimbus, which is like Hadoop's Job-Tracker. The worker node is known as Supervisor. It works when Nimbus assigns a specific process to it. Thus, each sub-process of a topology works with several distributed machines. Zookeeper performs the role of coordinator between Nimbus and the Supervisors. More significantly, if there is a failure in any cluster, another cluster is assigned. So, the slave node controls the execution of its own tasks.

Features:

- It is known for processing millions of byte data per second per node
- It has cluster of machines where parallel calculations run across it
- Apache Storm restart the process on another node, whenever an active node die.
- Storm assures that each unit of data will be processed at least once
- Storm is one of the coolest tools for Bigdata analysis once it is deployed

Pros: It is effortless to setup, high scalability, speed, fault tolerance, applicable for most of the languages

Cons include complex operation, debugging issues and not very learner-friendly

4.5.APACHE FLINK

Flink was a stratospherere project ,started in Technical university ,Berlin in 2009 which later was incubated to Apache in March,2014.Apache Flink is developed by the Apache Software Foundation which is an open source framework that is used effectively for processing data both in Real time and batch mode. A special focus for the developers is the high processing speed of the data and data streams: The processing takes place in real time and without intermediate storage of the collected data in separate databases.

APACHE Flink's core is a streaming dataflow engine that offers data distribution, communication, and fault tolerance for distributed computations over data streams. APACHE Flink develops a batch processing which is found over the streaming engine, covering native iteration support, managed memory, and program optimization.

It uses in-memory processing technique and provides several APIs such as stream processing API (data stream), batch processing API (data set), and table API that has been used for queries. It contains machine learning (ML) and graph processing (Gelly) libraries as well.

Apache Flink provides

1. Bounded data streams.

2. Unbounded data streams

Computations of Streams:

Unbounded streams have a start but without a definite end. They provide data as it is generated and never terminate the stream. Unbounded streams must be continuously processed, i.e., events must be handled right away after they have been ingested. It is impossible to wait for all input data to arrive because the input is unbounded and will be uncompleted at any point in time. Processing unbounded data often involves the events being ingested in a specific order, such as the order in which events occurred, to be able to make sense about result completeness.

Bounded streams have their start and end well defined. Bounded streams can be processed by ingesting all data prior to performing any computations. Ordered ingestion is not required to process bounded streams since a bounded data set can always be separated. Processing of bounded streams is also called batch processing.

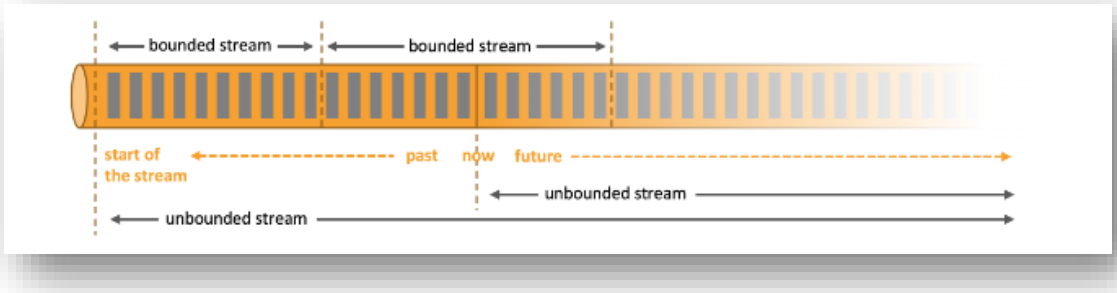

Fig 8: **Apache Flink Streams**

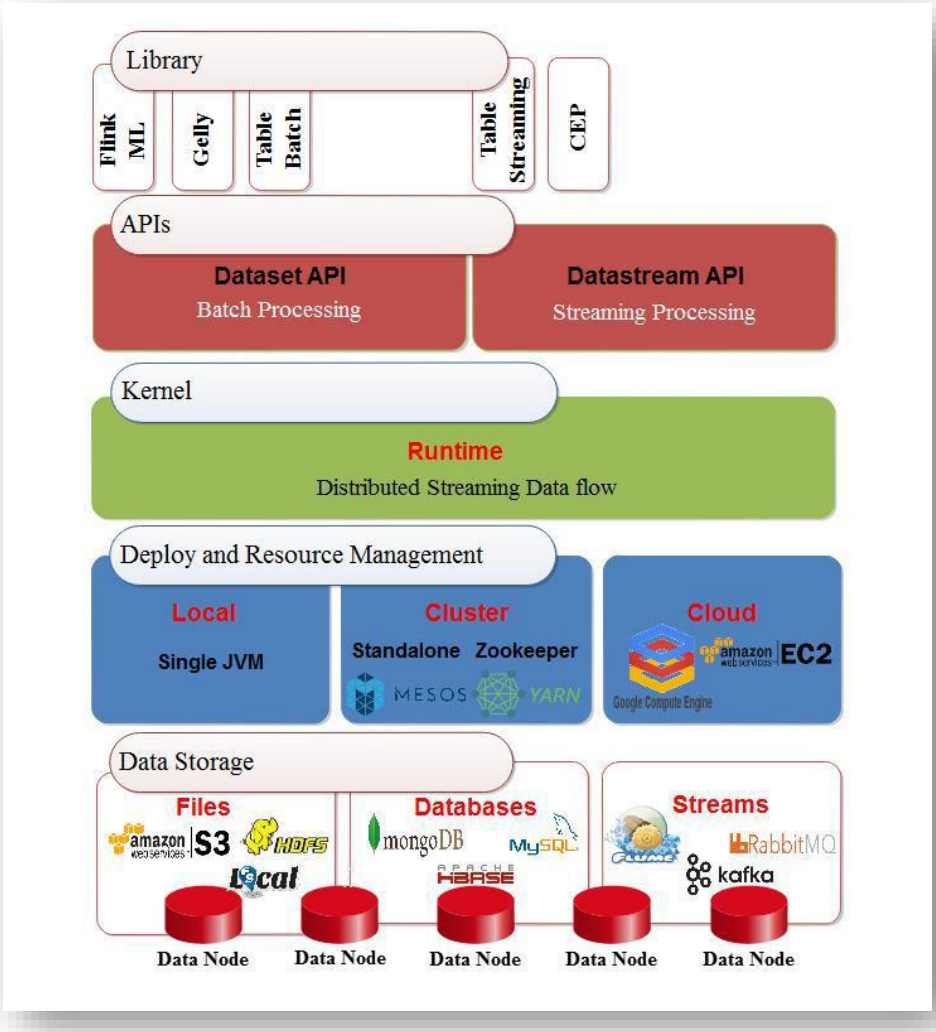

Fig 9: Apache Flink Architecture

The above figure illustrates the architecture of Flink. In the base layer, the storage layer can read and write the data from multiple destinations such as HDFS, local files, and so on. Then, the deployment and resource management layer contain the cluster manager for managing the

tasks of planning, monitoring the jobs, and managing the resources. This layer also contains the environment that executes the program, which are the clusters or cloud environments. Besides, it has the local area for single Java virtual machine.
Moreover, it has the kernel layer for distributed stream data flow engine for real-time processing. Also, the application program has interface layers for two processes: batch and streaming. The upper layer is a library in which the program is written in Java or Scala programming language**.** It is then sent to the compiler for conversion with the aid of the Flink optimizer in order to better its performance.

Apache Flink is a processing framework, which pays attention to the processing models and their exposure to different domains.
Storage: Flink does not have its own data storage. Flink does not care about abstractions of dealing with the storage because it works with all kinds of storage. Flink is data source agnostic. It supports several storage types such as local file, kafka, jdbc connection etc.
Deploy: Flink provides various execution environments such as Local, cluster, Yarn cloud etc.
Core: Flink supports both stream and batch processing.

API: Flink has the unique classes of DataSet and DataStream showing data in a program. Both are immutable collections of data that can contain duplicates. In case of datasets, the data is fixed, and it concentrates on batch processing of data while for a DataStream the number of components can be unbounded and it emphases on real time processing of data

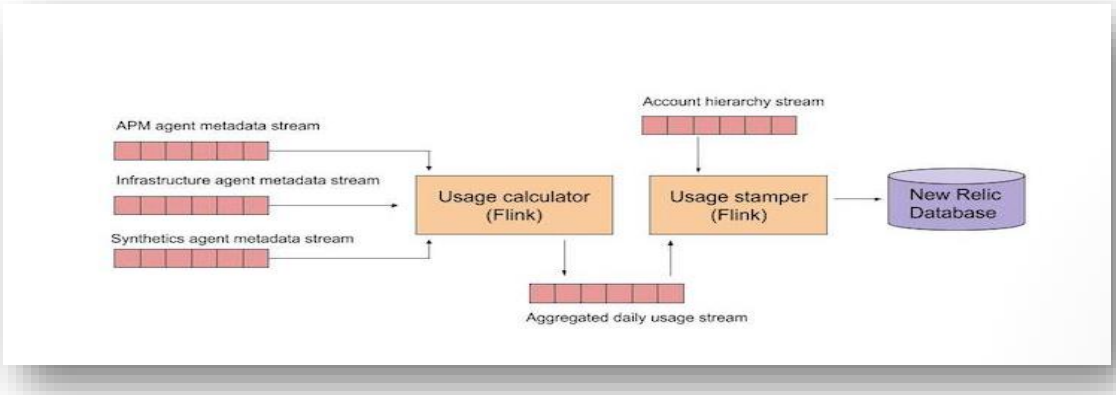

Fig 10: Apache Flink Architecture

Consists of Usage Calculator and the Usage Stamper. The Usage Calculator is an application that reads from Kafka topics comprising usage metadata from New Relic Infrastructure, New Relic APM and New Relic Synthetics agents; the application sums data for 24 hours and then writes that data to a Kafka topic holding daily usage data. The Usage Stamper reads from Kafka topics, and relates the usage data to its account order, which is derived from a distinct Kafka topic. Necessarily, every Flink app reads from a stream of input, runs a handful of operations in parallel to transform the data, and writes the data out to a datastore.

Writing code to achieve a basic Flink application running is unexpectedly simple and relatively concise. The application fits into the read, process, write model:

1. The app consumes data on Kafka topics from the agents.
2. The app runs the data through several operations to calculate the usage for each activity for each vehicle, using business rules specific to the vehicle.
3. The app writes the result to a Kafka topic that is read by the Usage Stamper app.

Features:

- Gives results that are correct, even for late-arriving data or out-of-order
- Fault-tolerant, stateful and it has reliable recovery from failures
- Performs at a large scale, running on several nodes
- Has good throughput and latency characteristics
- This tool supports windowing with event time semantics and stream processing
- Supports easily modifiable windowing based on time, count to data-driven windows
- It supports a wide range of connectors to third-party systems for data sources and sinks
- Generalized distributed data processing

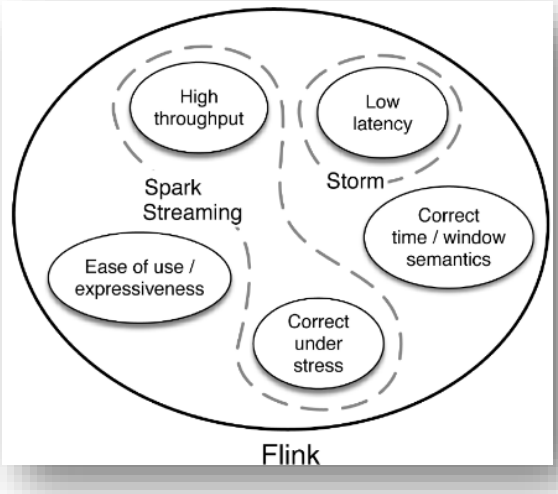

Fig 11: Apache Flink Components

Pros

Apache Flink helps make stream processing more efficient. By processing in real time, data streams can be processed as soon as they arise thus providing low latency. Flink no databases are necessary to temporarily store the data before analysis. Unlimited data streams can be processed on-site and at the time they are created.

There is still the option to process static data. If necessary, data can also be stored in a database and processed with Apache Flink.

Cons include few scalability issues.

4.6.APACHE SPARK

Apache Spark is an open source framework that was established at the University of California, Berkeley. It developed into an Apache project in 2013, providing faster services with large-scale data processing.

Apache Spark is developed for fast computations. It is an advanced computing technology based on Hadoop-MapReduce and it also extends the Map Reduce model in order to use it more efficiently for various forms of computations and streaming processes. It supports in memory-based cluster computing which yields the increase of the processing speed of an application.

The goal of developing Spark was to cover a wide range of data processing workloads such as batch processing, iterative algorithms, interactive queries and streaming of data. Apart from

supporting data processing in a respective system, it also decreases the heavy load of maintaining a separate tool for management.
The major features of spark are speed, support for multiple language and advanced analytics.

Spark framework is to Hadoop what MapReduce is to data processing and HDFS. In addition, Spark has data sharing known as Resilient Distributed Datasets (RDD) and Directed Acyclic Graph (DAG). Avoiding the combination of SI and CGS units, such as current in amperes and magnetic field in oersted. This usually leads to confusion since, equations are not balanced dimensionally. If you must use mixed units, the units for each quantity that you used in the equation must be plainly stated.

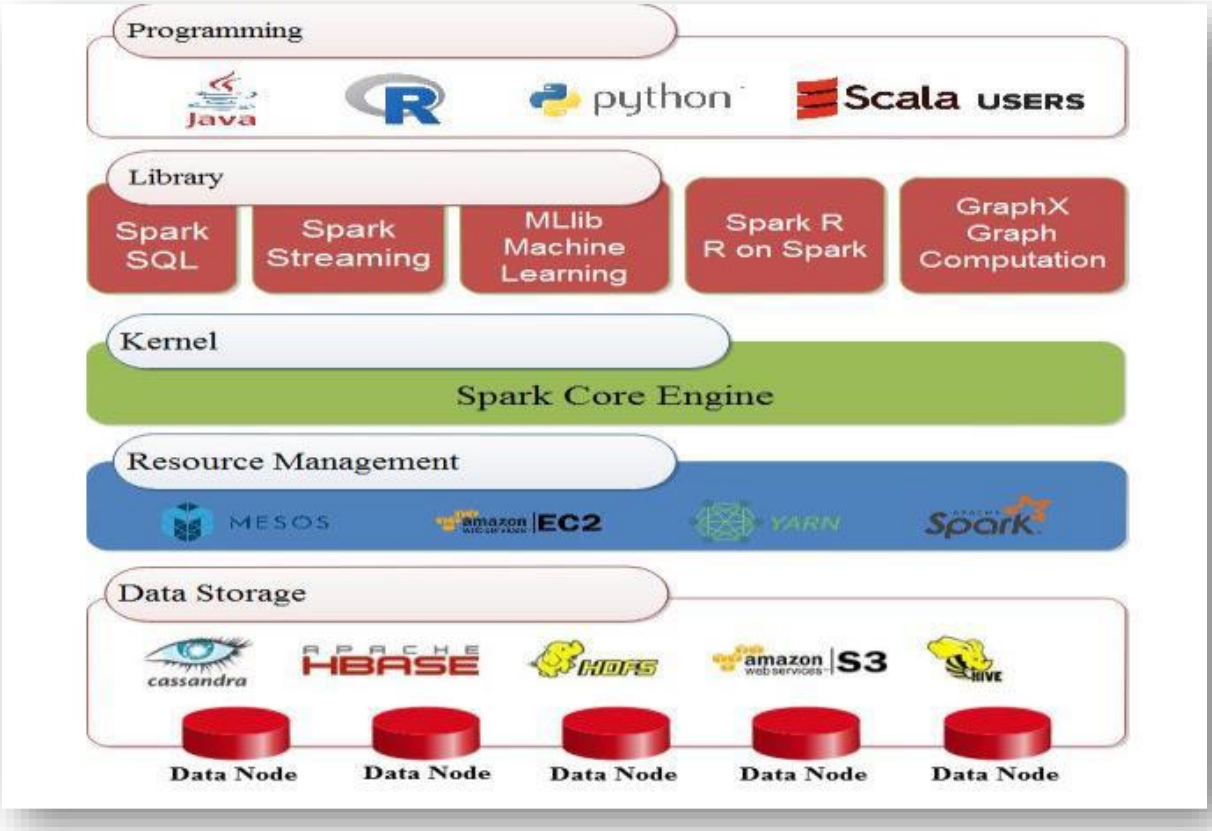

Fig 12: Apache Spark Components

The above figure shows spark architecture, which is easier and faster for selecting a large amount of data processing. Spark mostly comprises of five layers. The first layer consists of data storage systems such as HDFS and HBASE. The second layer is the resource management for instance, YARN and Mesos. Spark core engine is the third layer. The fourth layer is a library, which consists of SQL, stream processing, MLlib for machine learning, Spark R, and GraphX for graph processing. The final layer is an API such as Java or Scala.

Structure & features of Spark

- **Apache Spark Core-** It is general execution engine for spark platform on which each one of the alternate functionalities are made. It helps In-Memory computing and referencing datasets in memory device systems.
- **Spark SQL**-Spark SQL is a portion on top of the Spark's Core that introduces an alternative information abstraction named as SchemaRDD, that supports structured and semi-structured information.
- **Spark Streaming**-Spark Streaming controls Spark Core's quick programing capability to achieve streaming analytics. It consumes information in mini-batches and performs RDD (Resilient Distributed Datasets) transformations on those mini-batches of information.

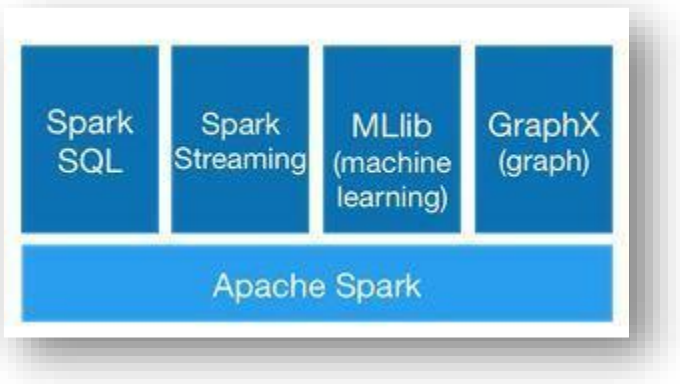

Fig 13: Apache Spark Components

- **Machine Learning Library -** Apart from spark's distributed memory-based design, it is a distributed ML framework. Spark MLlib is 9 times fast when compared to Hadoop disk-based version of Apache mahout.
- **GraphX-** Apart from spark's inbuilt API for expressing graph computation, it is a distributed graph-processing framework. Here it will model the user-defined graphs by usage of the Pregel abstraction API.

Benefits of Apache Spark

- Processing Time
- CPU Consumption
- Execution Time
- Latency
- Scalability

Pros: It includes scalability, incredibly quick by reduced number of input & output operations to disk, fault tolerance, assist advanced analytics applications with excellent Artificial Intelligence implementation and continuous integration with Hadoop

Cons: It is little complex in setup and implementation, has boundaries in language support.

5. FEATURE COMPARISON OF BIG DATA FRAMEWORKS

Each of the big data frameworks in our study supports a set of features, which could be used as a key performance indicator too. In this section, we will present a set of common features identified through literature review and compare the four frameworks across these features.

A. Scalability

Scalability is the ability of a system to respond to increasing amount of load. It has two types: scale up (vertically) and scale out (horizontally). Scale up is used to upgrade the hardware configuration, whereas scale out is used to add extra hardware. It means we can add many nodes to the cluster as and when required.

B. Message Delivery Guarantees

Message delivery guarantees are used in the case of failure. According to the four frameworks mentioned above, it can be divided into two types: exactly once delivery and at-least-once delivery. Exactly once delivery means that the message will not be duplicated, nor be lost, and will deliver to the recipient exactly once. On the other hand, at-least-once delivery means there are many attempts to deliver the message and at least one of these attempts succeeds. In addition, the message can be duplicated without being lost.

C. Computation Mode

Computation mode could be in-memory computing or the more “traditional” mode where computation results are written back to the disk. In-memory computing is faster but comes at a potential disadvantage of losing the contents in case of the machine being turned off.

D. Auto-Scaling

Auto-scaling refers to the automatic scaling of cloud services, depending on the situation it can be either up or down.

E. Iterative Computation

Iterative computation refers to the implementation of an iterative method that evaluates an approximate solution in the absence of a real solution or when the cost of a real solution is excessively high.

TABLE 1: SUMMARIZATION OF SOME FEATURES OF BIG DATA FRAMEWORKS

Features	***Hadoop***	***Spark***	***Storm***	***Flink***
Processing Mode	Batch	Batch and Stream	Stream	Batch and Stream
Scalability	Horizontal	Horizontal	Horizontal	Horizontal
Message Delivery Guarantees	Exactly once	Exactly once	At least once	Exactly once
Computation Mode	Disk-based	In memory	In memory	In memory
Auto-scaling	Yes	Yes	No	No
Iterative Computation	Yes	Yes	Yes	Yes

TABLE 2: COMPARISON OF BIG DATA FRAMEWORKS WITH BEST FEATURES

Categorized	**In case of**	**Hadoop**	**Spark**	**Flink**	**Storm**
Processing time	Big data set	Less faster	Fastest	Slower	Not Compared
Processing time	Small data set	Slower	Fastest	Less faster	Not Compared
Processing time	Cluster size	Slow	Fast	Slow	Not Compared
Processing time	JSON format data	Not Compared	Fast	Slow	Not Compared
CPU consumpt-ion	Stream mode	Not Compared	Less CPU usage	Less CPU usage	Highest CPU Usage
Execution time	DAS-4 and Tera Sort	High execution time	Low execution time	Low execution time	Not Compared
Execution time	Word Count and logistic regress-ion program	High execution time	Low execution time	Not Compared	Not Compared
Execution time	Word Count	High execution time	Low execution time	Not Compared	Not Compared
Scalability	Large dataset and fixed	Not Compared	Better	Best	Not Compared

6. Implementation:

Environmental Setup:

As per the process, we have setup the environment for the data processing and deployment in cloud.

Tools & Components Used:

Data streaming required components & elements are mentioned below,

- Spark V 3.0.3
- Winutils
- Eclipse IDE
- Scala Plugin latest version for Eclipse

We have configured the above components for the data streaming implementation locally and in cloud (Amazon AWS).

Cloud:

Created new account with Amazon AWS and initiated an EC2 Instance for deploying the Data Streaming process using Spark + Scala

- AWS Account
- Root User
- EC2 Instance
- Windows Server

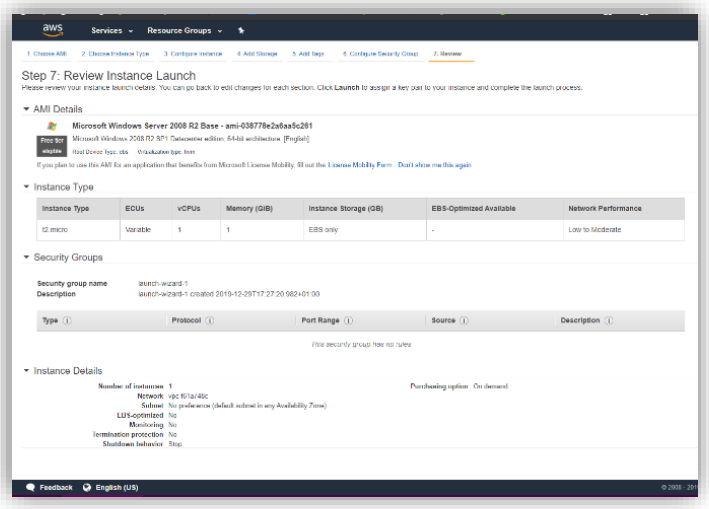

Fig 14: AWS Instance

Windows Server in AWS Cloud:

AWS allocated the windows server which as shown below,

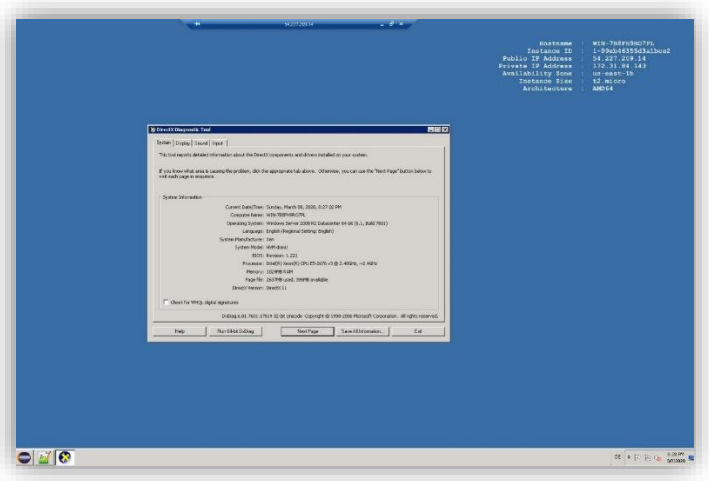

Fig 14: AWS Remote Windows Server

Spark & Scala in AWS Cloud:

- We have installed Spark V 3.0.0, Eclipse IDE, Scala Plugin using Eclipse Market Place to integrate Apache Spark + Scala + Eclipse IDE.

Uber Data Analysis:

Problem Statement: Finding the days on which each basement has received more trips in Uber.

- As per our project we have added the data file with uber trips information. We have reviewed the data as per our plan with different headers and details.
- After integrating the **Scala plugin + Eclipse IDE** we have created the package and project with "**Scala Nature**"
- Created "**Scala Object**" for adding required Scala coding as per our uber data
- Spark dependencies/libraries has been added to the into the project

Process: (Cloud environment)

Initially, the data has been fetched into Eclipse IDE from the file stored in cloud server. We have assigned variables for assigning & differentiating the headers of our data, also the date formats since we are using the date which sorted out using dates. We have used import classes for date formats.

We have used arrays to define the weekdays.

The major functionalities we focused are **(MAP - REDUCE - FILTER)**

FILTER:

- We are filtering the headers in our data to get the required data as per our state.
- We used "**filter**" function to return the new dataset as per the header we defined, it filters the data from the source (Data) which we fetched into it.

MAP:

- Used **MAP** function to map the elements of the data using array index.
- It applied each element of **RDD** and given the defined data as per the index

REDUCE:

- We have added Spark RDD **reduceByKey** function which merged the values for each key using an associative reduce function.
- Added this function since, it can do transformation operation that performs aggregation of data. It receives key-value pairs (K, V) as an input, aggregates the values based on the key and generates a dataset of (K, V) pairs as an output.

Overall, we sorted out the days from the dataset (data file) when the basement had a greater number of trips.

7. Conclusion

Overall, as per our research review on the big data streaming frameworks, the features which we have considered for the analysis are the essential areas required for the best outcome for the data stream processing. From TABLE 1, we considered the following features: Processing Mode, Scalability, Message Delivery Guarantees, Computation Mode, Auto-Scaling, Iterative Computation. Additionally, we took each feature and had a deep dive into them with respect to their benchmarks. The processing time, CPU-consumption, Scalability, Execution Time were analyzed in our project, where spark framework has proven to be best on those above-mentioned criteria during our project implementation.

After our analysis we found there is no clear-cut framework that suits best for all business scenarios in big data processing. In this paper, we highlighted few frameworks, for example Apache Hadoop is compatible with most of the emerging technologies in big data services. With this advantage it might still not be suitable for the companies who prioritize security. When it comes to ability to process the batch & stream data simultaneously Apache Flink is

desirable to use. However, Apache Spark is relatively faster than Flink in terms of computation time. In cases, where there are less CPU resources Apache Storm is less appropriate due to its high CPU consumption when compared to Apache Spark. Hence, handlers from various companies, researchers and individuals who are really involved in working with framework, one can choose the appropriate framework, based on the key performance indicators they wish to use. As per overall comparison with other frameworks taking main criteria into consideration, Spark would be the best choice to analyze data and gain efficient results.

References

1) Abdul Ghaffar Shoro & Tariq Rahim Soomro. Big Data Analysis: Apache Spark Perspective. Global Journal of Computer Science and Technology: C Software & Data Engineering. 2015. https://www.computerresearch.org/index.php/computer/article/view/1137

2) Muhammad Hussain Iqbal, Tariq Rahim Soomro. Big Data Analysis: Apache Storm Perspective. International Journal of Computer Trends and Technology (IJCTT) – Volume 19 Number 1 – Jan 2015. https://www.researchgate.net/profile/Tariq_Soomro/publication/271196175_Big_Data_Analysis_Apache_Storm_Perspective/links/54bff9900cf28a6324a02e6b.pdf

3) Monika Chand, Chetna Shakya, Gagandeep Singh Saggu, Deepesh Saha, Inish Krishna Shreshtha, Ankur Saxena. Analysis of Big Data using Apache Spark. IEEE Conference ID: 40353 2017 4th International Conference on "Computing for Sustainable Global Develop ment", 01st - 03rd March, 2017. http://studentlearning.in/wp-content/uploads/2018/04/Analysis-of-Big-Data-using-Apache-Spark-2.pdf

4) Ahmed Oussous, Fatima-Zahra Benjelloun, Ayoub Ait Lahcen, Samir Belfkih. Big Data technologies: A survey. Journal of King Saud University – Computer and Information Sciences 30 (2018) 431–448. https://www.sciencedirect.com/science/article/pii/S1319157817300034?via%3Dihub

5) Elkhan Shahverdi, Ahmed Awad and Sherif Sakr. Big Stream Processing Systems, An Experimental Evaluation. 35th International Conference on Data Engineering Workshops (ICDEW), IEEE 2019. https://ieeexplore.ieee.org/abstract/document/8750955

6) Karan Patel, Yash Sakaria and Chetashri Bhadane. Real Time Data Processing Frameworks, International Journal of Data Mining & Knowledge Management Process (IJDKP) Vol.5, No.5, September 2015. https://www.researchgate.net/publication/282776889_Real_Time_Data_Processing_Framework

7) Safaa Alkatheri, Samah Anwar Abbas, Muazzam Ahmed Siddiqui. A Comparative Study of Big Data Frameworks. International Journal of Computer Science and Information Security (IJCSIS), Vol. 17, No. 1, January 2019. https://www.researchgate.net/publication/331318859_A_Comparative_Study_of_Big_Data_Frameworks

8) Dr. Urmila R. Pol. Big Data Analysis: Comparison of Hadoop MapReduce and Apache Spark. International Journal of Engineering Science and Computing, June 2016. DOI 10.4010/2016.1535 ISSN 2321 3361 © 2016 IJESC. https://www.researchgate.net/profile/Urmila_Pol/publication/308074396_Big_Data_Analysis_Comparision_of_Hadoop_MapReduce_and_Apache_Spark/links/57d8fc4b08ae5f03b4986fd9/Big-Data-Analysis-Comparision-of-Hadoop-MapReduce-and-Apache-Spark.pdf

9) Carl Steinbach, Apache Hive, Apache Hadoop Community Spotlight, MARCH 2013. https://www.intel.ch/content/dam/www/public/us/en/documents/case-studies/big-data-apache-hadoop-framework-spotlight.pdf

10) Krati Bansal, Priyanka Chawla and Pratik Kurle, Analyzing Performance of Apache Pig and Apache Hive with Hadoop. Springer Nature Singapore Pte Ltd. 2019. https://link.springer.com/chapter/10.1007/978-981-13-1642-5_4

11) Telmo da Silva Morais. Survey on Frameworks for Distributed Computing: Hadoop, Spark and Storm. Proceedings of the 10th Doctoral Symposium in Informatics Engineering - DSIE'2015. https://paginas.fe.up.pt/~prodei/dsie15/web/papers/dsie15_submission_7.pdf

12) http://hadoop.apache.org/

13) http://www.vldb.org/pvldb/vol10/p1634-noghabi.pdf

14) http://martin.kleppmann.com/papers/samza-encyclopedia.pdf

15) https://www.alooma.com/blog/what-is-data-streaming

16) https://www.techopedia.com/definition/31747/real-time-data-streaming

17) https://www.algoworks.com/blog/real-time-data-streaming-tools-and-technologies/

18) https://www.thewindowsclub.com/cloud-computing-services

19) https://www.upsolver.com/blog/popular-stream-processing-frameworks-compared

20) https://jelvix.com/blog/top-5-big-data-frameworks

21) https://www.guru99.com/big-data-tools.html

22) https://www.cuelogic.com/blog/big-data-frameworks

23) http://nathanmarz.com/blog/history-of-apache-storm-and-lessons-learned.html