| Bsef19m001 | Atif | Spark, Storm |
| --- | --- | --- |
| Bsef19m010 | Nimra | DataBases |
| Bsef19m038 | Usman | Spark, Flink |
| Bsef19m039 | Ahmad | Kafka, Flume Document Proofread |

**Comparison of Different Open Source Tools for The Implementation of ETL PipeLine**

There are a variety of tools available for implementing ETL pipeline. So in this document we will be comparing thosetools and will discuss the reasons why we selected those tools.

**Open-Source Data Stream Processing Engines**

The stream processing engine organizes data events arriving in short batches and presents them to other applications as a continuous feed.

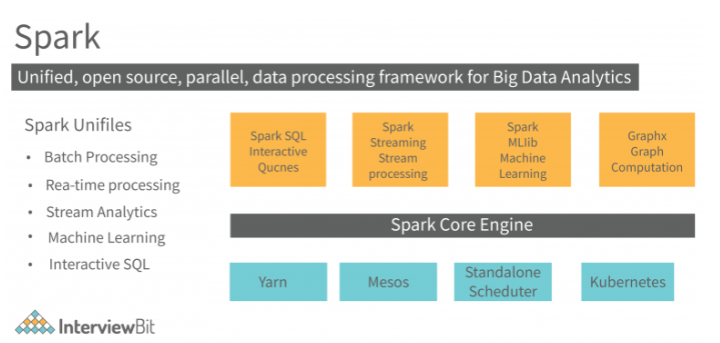
There are several data streaming processes engines. Some of them are Apache Kafka, Apache Spark, Apache Flume, Apache Flink, Apache Storm. These tools are discussed below:

**Apache Spark:**

Apache Spark is an open source data processing engine to store and process data in real time across various clusters of computers using simple programming constructors.

If you have a very large dataset that can represent anything, weblogs, genomics data, you name it.Spark can slice and dice that data up, it can distribute the processing amongst a huge cluster of computers and take a data analysis problem that's just too big to run on one machine and divide it and conquer it by splitting it up amongst multiple machines.And the way that it scales it is it can run on top of a cluster manager so your actual Spark scripts are just everyday scripts written in Python or Java or Scala and they behave just like any other script,your driver program is what we call it and it will run on your desktop or on one master node of your cluster, and it behaves just like any other script,Spark knows how to take the work and actually farm it out to different computers on your cluster or different CPUs on your same machine even.So Spark can actually run on top of different cluster managers,it has its own built in cluster manager that you can use by default but if you have access to a Hadoop cluster, there's a component of Hadoop called Yarn that Sparkalso run on top of to distribute work amongst a huge Hadoop clusterAnd that cluster manager will split up the work and coordinate among various executors.

So Spark will split up and create multiple executors per machine,Ideally you want one per CPU core, and it can do all the coordination using a cluster manager and also your driver program itself to farm out work and distribute it to different nodes and also give you fall tolerance,So if one of your executors goes down, it can recover without actually stopping your entire job and making you start it all over.So the beauty of it is that it scales out to entire clusters of computers, it gives you horizontal partitioning,horizontal scalability with basically the sky's the limit.But from a user standpoint, from a developer standpoint, it's all just one simple little program running on one computer that feels a lot like writing any other script,so it's kind of a nice aspect of Spark.



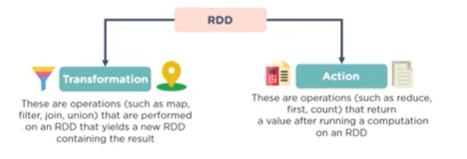
Components of Spark

1. Spark Core
2. Spark SQL
3. Spark Streamings
4. Sparks MLlib
5. Spark GraphX
6. Spark Core

It is a base engine for large scale parallel and distributed data processing. It is responsible for memory management, fault tolerance, scheduling, distribution and monitoring jobs on clusters, interaction with storage systems (Spark itself does not have its own storage it realizes on the storage like No sql, HDFS).

RDD

Spark core is embedded with RDDs (Resilient Distributed DataSets), an immutable fault tolerant, distributed collections of objects that can be operated on in parallel. It is a key tool for data computation. It enables you to recheck data in the event of a failure, and it acts as an interface for immutable data. It helps in recomputing data in case of failures, and it is a data structure. There are two methods for modifying RDDs: transformations and actions.



Rdd uses lazy evaluation i.e Nothing will run on your driver program until action is called.RDD's are both distributed and resilient, you know they can be spread out across an entire cluster of computers that may or may not be running locally and they can also handle the failure of specific executor nodes in your cluster automatically and keep on going even if one node shuts down and redistribute the work as needed when that occurs.It represents a really big data set and you can use the RDD object to transform that data set from one set of data to another, or to perform actions on that data set to actually get the results you want from it.

RDD Transformation:

There are some transforming method that you can perform on a RDD object:

1)Map 2)Flat Map

3)Filter 4)Distinct

5)Sample 6)Union,Intersection,Subtract,Cartesian

RDD Actions:

There are following actions that you can perform to get some useful information from dataset:

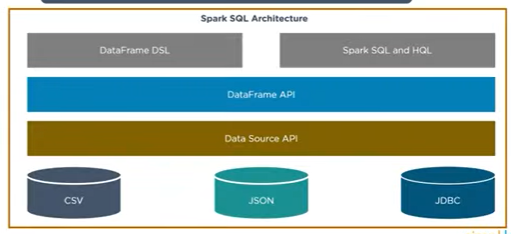
1)Collect 2)Count

3)Count By Value 4)Take

1. Spark SQL

It is a framework component used for structured and semi structured data processing.

Spark SQL Architecture:



1. Spark Streaming

Spark Streaming is a lightweight API that allows developers to perform batch processing and real-time streaming of data with ease.

Provides secure, reliable, and fast processing of live data streams.



1. Spark MLlib

MLlib is a low-level machine learning library that is simple to use, is scalable, and compatible with various programming languages.

MLlib eases the deployment and development of scalable machine learning algorithms. It contains machine learning libraries that have an implementation of various machine learning algorithms i.e. Clustering, Classifications, Collaborative Filtering

1. Spark GraphX

GraphX is Spark’s own Graph Computation Engine and data store

***Advantages of Spark:***

1. Speed – Spark can execute batch processing jobs 10 to 100 times [faster than MapReduce](https://www.projectpro.io/article/hadoop-mapreduce-vs-apache-spark-who-wins-the-battle/83). That doesn’t mean it lags behind when data has to be written to (and fetched from) disk, as it is the world record holder for large-scale on-disk sorting.
2. Ease of Use – Apache Sparkhas easy to use APIs, built for operating on large datasets.
3. Unified Engine – Spark can run on top of Hadoop, making use of its cluster manager (YARN) and underlying storage (HDFS, HBase, etc.). However, it can also run independent of Hadoop, joining hands with other cluster managers and storage platforms (the likes of Cassandra and Amazon S3). It also comes with higher – level libraries that support [SQL](https://www.projectpro.io/article/nosql-vs-sql-4-reasons-why-nosql-is-better-for-big-data-applications/86) queries, data streaming, machine learning and graph processing.
4. Choose from Java, Scala or Python – Spark doesn’t tie you down to a particular language and lets you choose from the popular ones such as Java, Scala, Python, R and even Clojure.
5. In-memory data sharing – Different jobs can share data within the memory, which makes it an ideal choice for iterative, interactive and event stream processing tasks.
6. Spark Fault Tolerance is the ability of a system to continue to function properly even if some of its components fail (or have one or more faults within them).
7. Active, expanding user community – An active user community has led to a stable release of Spark (in June, 2016) within 2 years of its initial release. This speaks volumes of its worldwide acceptability, which is on the rise.

**Apache Flink:**

Flink is an open-source framework for stateful, large-scale, distributed, and fault-tolerant stream processing.

Apache Flink allows to ingest massive streaming data (up to several terabytes) from different sources and process it in a distributed fashion way across multiple nodes, before pushing the derived streams to other services or applications such as Apache Kafka, DBs, and Elastic search. Simply, the basics building blocks of a Flink pipeline: input, processing, and output. Its runtime supports low-latency processing at extremely high throughputs in a fault-tolerant manner. Flink capabilities enable real-time insights from streaming data and event-based capabilities. Flink enables real-time data analytics on streaming data and fits well for continuous Extract-transform-load (ETL) pipelines on streaming data and for event-driven applications as well.

It gives processing models for both streaming and batch data, where the batch processing model is treated as a special case of the streaming one (i.e., finite stream). Flink’s software stack includes the DataStream and DataSet APIs for processing infinite and finite data, respectively. Flink offers multiple operations on data streams or sets such as mapping, filtering, grouping, updating state, joining, defining windows, and aggregating.The two main data abstractions of Flink are DataStream and DataSet, they represent read-only collections of data elements. The list of elements is bounded (i.e., finite) in DataSet, while it is unbounded (i.e., infinite) in the case of DataStream.

Flink programs are represented by a data-flow graph (i.e., directed acyclic graph — DAG) that gets executed on the Flink’s core, which is a distributed streaming dataflow engine. The data flow graphs are composed of stateful operators and intermediate data stream partitions. The execution of each operator is handled by multiple parallel instances whose number is determined by the parallelism level. Each parallel operator instance is executed in an independent task slot on a machine within a cluster of computers. The figure below shows an example of the data flow graph for Flink’s application

The following is a brief description of the main features of Flink:

* Robust Stateful Stream Processing: Flink applications give the ability to handle business logic that requires a contextual state while processing the data streams using its [DataStream API](https://ci.apache.org/projects/flink/flink-docs-release-1.9/dev/datastream_api.html) at any scale
* Fault Tolerance: Flink offers a mechanism of state recovery from faults based on a periodic and asynchronous checkpointing (saving internal state to external persistent storage such as HDFS)
* Exactly-Once Consistency Semantics: the Flink’s core ensures that each event in the stream is delivered and processed exactly once in case of failures
* Scalability: applications are parallelized to scale in or out the number of the processing tasks
* In-Memory Performance: Flink applications perform all computations by accessing local, often in-memory, state yielding very low processing latencies
* Flink provides seamless connectivity to a variety of data sources including Apache Kafka, Elasticsearch, Apache Cassandra, Kinesis and … many more
* Flexible deployment: Flink can be deployed on various clusters environments s such as YARN, Apache Mesos, and Kubernetes
* A Complex Event Processing (CEP ) library to detect patterns (i.e., event sequences) in data streams
* Flink is a true streaming engine comparing for instance to the micro-batch processing model of Spark Streaming

**Apache Storm:**

Apache Storm is a distributed stream processing computation framework written in the Clojure programming language. Originally created by Nathan Marz and team at BackType, the project was open sourced after being acquired by Twitter. The initial release was on 17 September 2011.

Apache storm is a real-time big data-processing system. Storm is designed to process vast amounts of data in a fault-tolerant and horizontal scalable manner. It is a streaming data framework that has the capability of highest ingestion rates.

**Use Cases of Apache Storm**

Apache storm is particularly useful for real time big data processing. For this reason, most of the companies are using Storm as an integral part of their system. Some notable examples are as follows:

* Twitter is using Apache Storm for its range of “Publisher Analytics Products”. “Publisher Analytics Products” process each tweet and clicks in the Twitter platform.
* NaviSite is using Storm for the Event log monitoring/auditing system. Every log generated in the system will go through the Storm.
* Spotify uses Storm for various real-time features, such as monitoring, analytics, recommendation systems, and targeting. With other technologies, such as Kafka and Cassandra.

**Benefits of Apache Storm:**

* Storm is open source, robust, and user friendly. It could be utilized in small companies as well as large corporations.
* Storm is fault tolerant, flexible, reliable, and supports any programming language.
* Allows real-time stream processing.
* Storm is unbelievably fast because it has enormous power of processing data.
* Storm can keep up the performance even under increasing load by adding resources linearly. It is highly scalable.
* Storm performs data refresh and end-to-end delivery response in seconds or minutes depending upon the problem. It has extremely low latency.
* Storm has operational intelligence.
* Storm provides guaranteed data processing even if any of the connected nodes in the cluster die or messages are lost.

**Disadvantages of Apache Storm**

* It is tricky to install and configure for deployment. The system integrates with various other technologies.
* No framework-level support. Project development starts from nothing, making it difficult for new developers to pick up.
* Not suitable for smaller datasets.

**Apache Kafka:**

Apache Kafka is a distributed [data streaming](https://www.redhat.com/en/topics/integration/what-is-streaming-data) platform that can publish, subscribe to, store, and process streams of records in real time. It is designed to handle data streams from multiple sources and deliver them to multiple consumers. In short, it moves massive amounts of data—not just from point A to B, but from points A to Z and anywhere else you need, all at the same time.

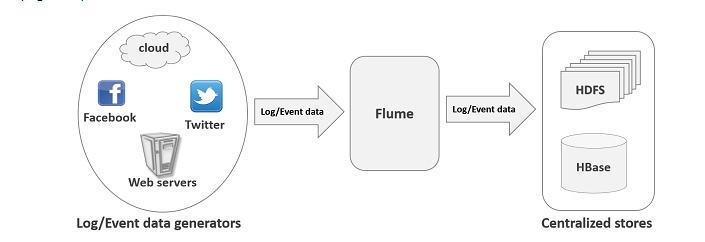
Apache Kafka is an alternative to a traditional enterprise messaging system. It started out as an internal system developed by Linkedin to handle 1.4 trillion messages per day, but now it's an open source data streaming solution with applications for a variety of enterprise needs.

Apache Kafka is built into streaming data pipelines that share data between systems and/or applications, and it is also built into the systems and applications that consume that data. Apache Kafka supports a range of use cases where high throughput and scalability are vital. Since Apache Kafka minimizes the need for point-to-point integrations for data sharing in certain applications, it can reduce latency to milliseconds. This means data is available to users faster, which can be advantageous in use cases that require real-time data availability, such as IT operations and e-commerce.

Apache Kafka can handle millions of data points per second, which makes it well-suited for [big data](https://www.redhat.com/en/insights/big-data) challenges. However, Kafka also makes sense for companies that are not currently handling such extreme data scenarios. In many data processing use cases, such as the [Internet of Things](https://www.redhat.com/en/topics/internet-of-things/what-is-iot) (IoT) and social media, data is increasing exponentially, and may quickly overwhelm an application you are building based on today's data volume. In terms of data processing, you must consider scalability, and that means planning for the increased proliferation of your data.

**Apache Flume:**

Apache Flume is a tool/service/data ingestion mechanism for collecting, aggregating and transporting large amounts of streaming data such as log files, events (etc...) from various sources to a centralized data store. Flume is a highly reliable, distributed, and configurable tool. It is principally designed to copy streaming data (log data) from various web servers to HDFS.



## Advantages of Flume:

Here are the advantages of using Flume −

* Using Apache Flume we can store the data into any of the centralized stores (HBase, HDFS).
* When the rate of incoming data exceeds the rate at which data can be written to the destination, Flume acts as a mediator between data producers and the centralized stores and provides a steady flow of data between them.
* Flume provides the feature of contextual routing.
* The transactions in Flume are channel-based where two transactions (one sender and one receiver) are maintained for each message. It guarantees reliable message delivery.
* Flume is reliable, fault tolerant, scalable, manageable, and customizable.

## Limitations of Apache Flume:

Some of the limitations of Apache Flume are:

### **Weak ordering guarantee**

Apache Flume offers weaker guarantees than the other systems such as message queues in the event of moving data more quickly and for enabling cheaper fault tolerance. In Apache Flume’s end-to-end reliability mode, the flume events are delivered at least once, but with zero ordering guarantees.

### **Duplicacy**

Apache Flume does not guarantee that the messages reaching are 100% unique. In many scenarios, the duplicate messages might pop in.

### **Low scalability**

Flume scalability is often low because for any businesses, sizing the hardware of a typical Apache Flume may be tricky, and in most of the cases, it is trial and error. Due to this, Flume's scalability aspect is often under the lens.

### **Reliability issue**

The throughput that Apache Flume can handle depends highly upon the backing store of the channel. So, if the backing store is not chosen wisely, then there may be scalability and reliability issues.

### **Complex topology**

It has complex topology and reconfiguration is challenging.

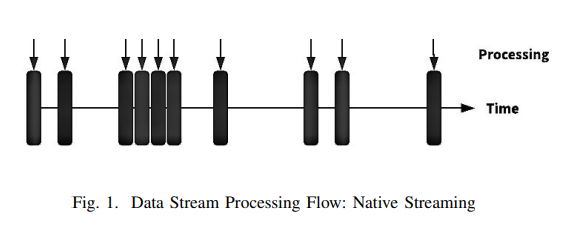
Despite its disadvantages, Flume’s advantages outweigh its disadvantages.

**Comparing Apache Spark and Apache Flink:**

**1)Processing Architecture:**

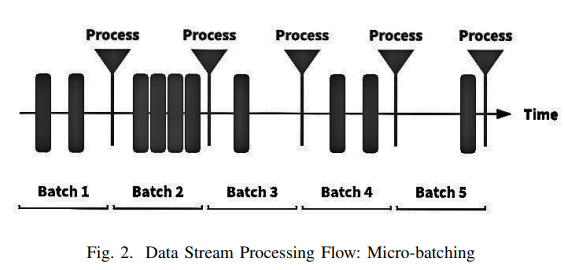
Spark Streaming engine is based on the concept of micro-batching while Flink and Storm are a native streaming engine.

A.**Native Streaming**:The native streaming models are designed to take into account the need of real-time applications, Fig. (1) demonstrates the processing of data streams obtained from producer sources over time, which is processed individually on an ongoing basis. This architecture helps to decrease latency owing to decreased waiting time before it gets into the system. Apache Flink and Storm with directed graph data flow adheres to this architecture resulting in reduced latency relative to micro batching model oriented Apache Spark.



As the stream data is handled separately and not in batch, this results in lower throughput compared to the micro batching. However, different studies showing configurable back-end implementation approaches to better handle tradeoffs to satisfy streaming application requirements.

**B. Micro-Batching:**



Micro-batching based architecture takes a continuous input data stream from multiple sources as shown in Fig. (2) and splits the stream into small batches or groups. Sets of those batches are then parallelly processed at tiny time intervals by processing engines. Apache Spark Streaming follows such architecture for managing streams in small batches. All the sources and stream processing nodes to gather creates a Directed Acyclic Graph (DAG). At the core of Apache Spark, batches are processed following this model as Discrete Streams (D-Stream) made of Resilient Distributed Datasets (RDDs).

2**)Fault-Tolerance:** As the system is vulnerable to failures owing to network or software errors, it is of primary importance in streaming applications. Spark streaming uses fault tolerance mechanisms for individual batches while it is expensive for a native streaming system such as Storm and Flink as it is enforced at each record level. Spark provides assurance of Exactly-Once’ processing of records in case of failures by continuously replicates state to the other worker nodes so that in the failure state can be extracted from other node and processing can be restarted. [10] [6] Similarly, Flink also provides ’Exactly-Once’ processing assurance by keeping track of distributed snapshots and checkpoints to provide failure recovery. [9] [6] [2] however, Storm does not provide state management and in case of application failure, it restarts the entire process again on different nodes giving ’At-leastOnce’ assurance. [3] [6] Exactly-Once’ processing of records in case of failures by continuously replicates state to the other worker nodes so that in the failure state can be extracted from other node and processing can be restarted. [10] [6] Similarly, Flink also provides ’Exactly-Once’ processing assurance by keeping track of distributed snapshots and checkpoints to provide failure recovery. [9] [6] [2] however, Storm does not provide state management and in case of application failure, it restarts the entire process again on different nodes giving ’At-leastOnce’ assurance.

**3)State-Management:** To manage the state a separate thread is required to continuously update and preserve the existing state of records.State-management in Spark streaming is associated with RDDs and involves updating each batch despite no change in the state, which makes it extremely inefficient compared to Flink. [10] Flink provides efficient support for integrating state management with the help of a distributed file system to keep track of state with snapshots.

**4)Performance (Latency Vs Throughput):** Latency is a time, records in the stream have to wait after it is produced and throughput is the number of records being processed by the system at a given unit of time. Studies show Spark streaming micro-batching model leads to higher latency and high throughput whereas Storm and Flink like native streaming platforms continuously process those records giving low latency.Spark works best with high throughput when the incoming volume is huge and latency is not of priority.

**5)Nature of Data:**If a stream contains skewed data, then Spark is the best choice (Experiment 4). Both Flink and Spark are very robust to fluctuations in the data arrival rate in aggregation workloads (Experiment 5). For fluctuations in the data arrival rate on join queries, on the other hand, Flink behaves better (Experiment 5).

**6)Optimizations** are automatically built-in to Flink. Spark batch and iterative jobs have to be manually optimized and adapted to specific datasets through fine grain control of partitioning and caching.

**Apache Storm vs Apache Spark**

| **Apache Storm** | **Apache Spark** |
| --- | --- |
| **Processing Model** | |
| Apache Storm supports true stream micro-batch processing through the core Storm layer. | Spark supports batch processing, and Spark streaming is a wrapper over Spark batch processing. |
| **Messaging** | |
| It uses ZeroMQ, Netty framework for messaging. | It uses Akka, Netty framework for messaging |
| **Programming Language** | |
| Broader language support | It supports only Java, Scala, Python and R |
| **Fault Tolerance** | |
| The supervisor process restarts automatically when a process fails. The state management is managed by Zookeeper in Apache Storm. | In Spark, if a process fails, the work is restarted through its standalone process manager or Mesos and Yarn. |
| **Latency** | |
| Low latency | High Latency |
| **Throughput** | |
| Storm has a lower throughput as compared to Spark as it serves only 10k records per node per sec. | Spark, on the other hand, has a high throughput and serves 100k records per node per sec. |
| **Sources** | |
| The source of stream processing in Storm is Spout. | Spark uses HDFS as a source for stream processing |
| **Ease of Operability** | |
| Storm can get tricky with installation and deployment. It is dependent on the Zookeeper cluster to coordinate with other clusters, store states, and statistics. | It, itself, is the basic framework for Spark streaming. Spark clusters can be easily maintained on YARN. |
| **Autoscaling** | |
| Apache Storm allows the configuration of initial parallelism at various topology levels. It also supports dynamic rebalancing. | The Apache Spark community is currently developing dynamic scaling. |
| **Community** | |
| Many big corporations are running Storm, pushing the boundaries for performance and scale. | Apache Spark streaming is a developing community and is thus limited in expertise when compared to Storm. |

**Apache Kafka vs Apache Flume**

| **Apache Kafka** | **Apache Flume** |
| --- | --- |
| Apache Kafka is a distributed data system. | Apache Flume is an available, reliable, and distributed system. |
| It is optimized for ingesting and processing streaming data in real-time. | It is efficiently collecting, aggregating and moving large amounts of log data from many different sources to a centralized data store. |
| It is basically working as a pull model. | It is basically working as a push model . |
| It is easy to scale. | It is not scalable in comparison with Kafka. |
| A fault-tolerant, efficient and scalable messaging system. | It is specially designed for Hadoop. |
| It supports automatic recovery if resilient to node failure. | You will lose events in the channel in case of flume-agent failure. |
| Kafka runs as a cluster which handles the incoming high volume data streams in the real time. | Flume is a tool to collect log data from distributed web servers. |
| Kafka will treat each topic partition as an ordered set of messages. | Flume can take in streaming data from multiple sources for storage and analysis which are used in Hadoop. |

**Conclusion:**

The reason we choose spark is

**1)efficient Fault Tolerance** as Spark streaming uses fault tolerance mechanisms for individual batches while it is expensive for a native streaming system such as Storm and Flink as it is enforced at each record level

**2)Spark** works best with high throughput when the incoming volume is huge and latency is not of priority.

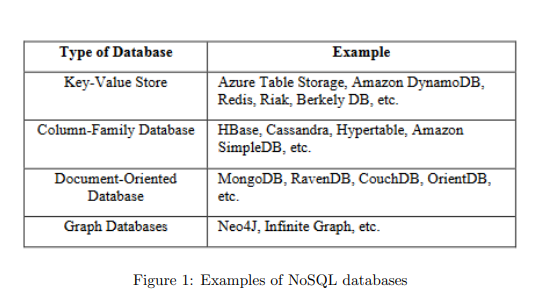
**3)Spark** is more mature than other engines. It has managed support and it is easy to find many existing use cases with best practices shared by other users. In some cases, you can even find existing open source projects to use as a starting point.

**4)**The usage and calling of API's in case of spark is easier than other

**Comparison of Different Databases**

One of the leading technical problems confronted by today’s companies is to ensure that a massive amount of data is stored, manipulated, and recovered efficiently. Online services and social media are the prime cause of data creation nowadays. Facebook generates 4 petabytes of data per day—that's a million gigabytes. Together, other social media apps, the internet of things (IoT), and geographic, vector space, electric power, wireless sensor, and power grids produce an immense amount of data, exceeding the scale of petabytes daily. Big data evolution is disclosing a severe systemic problem. A systemic problem is when the

The amount of data is increasing day by day, but it is not stored and processed efficiently. Conventional tools and techniques are unable to handle these complexities because data is growing in a huge volume and in terms of variety and substantial value, and includes different data types. It is necessary to handle such a huge amount of data and its significant problems efficiently.



Relational databases are an important scheme for storing data; however, they are not always the right choice for storing and querying data at scale. As NoSQL databases do not adhere to a strict schema, they can handle large volumes of structured, semi-structured, and unstructured data. Distributed databases are becoming increasingly popular to solve the problem of scalability and availability. Three such “big data” management systems are MongoDB, Cassandra, and HBase.

# **Apache Cassandra**

Apache Cassandra is an open source distributed database management system. It is an Apache Software Foundation top-level project designed to handle very large amounts of data spread out across many commodity servers while providing a highly available service with no single point of failure. It is a NoSQL solution that was initially developed by Facebook and powered their Inbox Search feature until late 2010.

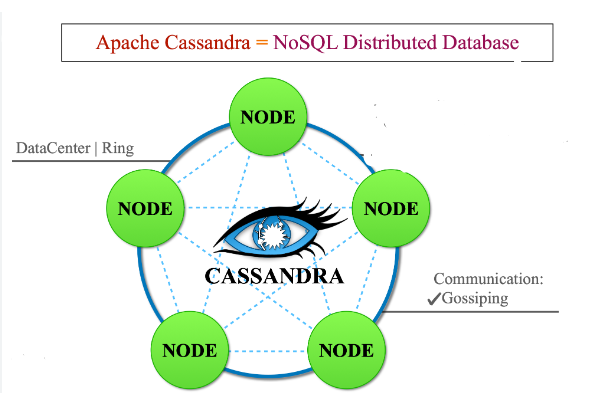
Cassandra provides a structured key-value store with tunable consistency. Keys map to multiple values, which are grouped into column families. The column families are fixed when a Cassandra database is created, but columns can be added to a family at any time. Furthermore, columns are added only to specified keys, so different keys can have different numbers of columns in any given family. The values from a column family for each key are stored together. This makes Cassandra a hybrid data management system between a column-oriented DBMS and a row-oriented store.

**Architecture**

The design goal of Cassandra is to handle big data workloads across multiple nodes without any single point of failure. Cassandra has a peer-to-peer distributed system across its nodes, and data is distributed among all the nodes in a cluster.

The nodes in cassandra have a similar role, all take care of reading and writing data, reducing the risk of bottleneck. Therefore, there are no dead spots. Furthermore, any number of servers / nodes can be added to any Cassandra cluster in any data center.

Nodes communicate with each other by Gossip communication – Gossip is a protocol used to update information about the state of other nodes participating in the cluster. This is a peer-to-peer communication protocol in which each node periodically exchanges its status information with the other nodes to which they are associated.



## **Features**

* **Decentralized :** Every node in the cluster has the same role. There is no single point of failure. Data is distributed across the cluster (so each node contains different data), but there is no master as every node can service any request.
* **Supports replication and multi data center replication :** Replication strategies are configurable. Cassandra is designed as a distributed system, for deployment of large numbers of nodes across multiple data centers. Key features of Cassandra’s distributed architecture are specifically tailored for multiple-data center deployment, for redundancy, for failover and disaster recovery.
* **Scalability :** Read and write throughput both increase linearly as new machines are added, with no downtime or interruption to applications.
* **Fault-tolerant :** Data is automatically replicated to multiple nodes for fault-tolerance. Replication across multiple data centers is supported. Failed nodes can be replaced with no downtime.
* **MapReduce support :** Cassandra has Hadoop integration, with MapReduce support. There is support also for Apache Pig and Apache Hive
* **Query language :** CQL (Cassandra Query Language) was introduced, an SQL-like alternative. The Cassandra Query Language (CQL) is the primary language for communicating with the Apache Cassandra database. The most basic way to interact with Apache Cassandra is using the CQL shell, cqlsh. Using cqlsh, you can create keyspaces and tables, insert and query tables, plus much more.

[1]

**Disadvantages**

* No join or subquery support. You may be able to find a workaround for this one, but that might affect the performance.
* Here data is modeled around queries instead of its structure due to which the same data is stored multiple times.

# **HBase**

Many large-scale applications in various business and scientific domains require both parallel computing and distributed data management for big data processing. HBase has been selected for several purposes, including its scalability, efficiency, and strong consistency support.

HBase is an open-source, distributed, multi-dimensional, and NoSQL database . It is designed to achieve high throughput and low latency. It provides fast and random read/write functionality on substantial datasets.The HBase runs on the top of HDFS and provides all capabilities of a large table to Hadoop. HBase stores files on HDFS. It has the capabilities to store and process a billion rows of data at a time.

HBase is a NoSQL and column-oriented database. While it looks like a relational database that includes rows and columns, HBase is not a relational database. It is a column-oriented database, while the relational databases are row-oriented.

**Features**

* HBase provides linear scalability and modularity to the database
* It offers consistent read/write operations
* It offers automatic sharding of tables and can also be configured as per user requirements
* In case of region servers, HBase supports automatic failover
* Real-time queries can be implemented with the help of Bloom Filters and Block caches
* Server side Filters can be used in HBase for Query predicate push down

**Disadvantages**

* Unlike Cassandra, HBase does not have a query language. This means that to achieve SQL-like capabilities, one must use the JRuby-based HBase shell and technologies like Apache Hive (which, in turn, is based on MapReduce). The major problem with this approach is the high latency and steep learning curve in employing these technologies.
* Although HBase scales well by adding DataNodes to its cluster, it has some high hardware requirements, mainly because of its dependency on HDFS, which would require five DataNodes and one NameNode as a minimum. This, in turn, translates to high running and maintenance costs.
* Another important factor to consider when choosing HBase is its interdependence on other systems, like HDFS, for storage, and Apache ZooKeeper for status management and metadata. So, when designing solutions, the architecture might become complex, and one must know these technologies well.
* Although HBase shares several similarities with Cassandra, one major difference in its architecture is the use of a master-slave architecture. This also proves to be a single point of failure, as failing from one HMaster to another can take time, which can also be a performance bottleneck. If you are looking for an always-available system, then Cassandra might be a better choice.

# **MangoDB**

MongoDB is an open source NoSQL database management system that works well with large sets of distributed data. MongoDB is a flexible document database that stores data as JSON files without a strict schema.

MongoDB documents store data in a series of key-value pairs. MongoDB documents are units of data called Binary JSON (BSON), which is like JSON . BSON can accommodate various data types, including documents, arrays, and arrays of documents. Documents incorporate a primary key as a unique identifier, which reflects the fact that MongoDB is somewhat relational . Sets of MongoDB documents form collections that can be viewed as relational database tables. Collections can contain any type of data but cannot be spread across different databases.

**Features**

* Can store various types of data, allowing users to create any number of fields in a document and making it more horizontally scalable than SQL databases. Horizontal scalability makes it a good partner for big data applications.
* MongoDB allows embedded documents, reducing the need for database joins and reducing cost.
* Sharding distributes data across a cluster of machines to perform distributed computing.
* MongoDB has official drivers for most programming languages and development platforms, as well as unofficial or open source drivers for other programming languages and platforms.
* In addition to supporting Hadoop, Spark, and other data processing frameworks, MongoDB has its own file system called GridFS, which is like Hadoop’s Distributed File System. [5]

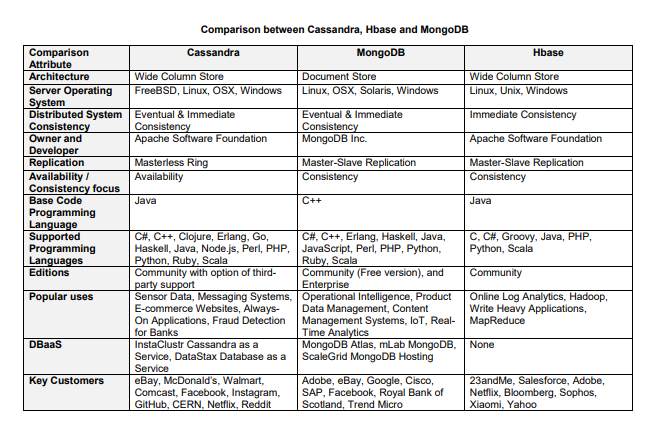
**Disadvantages**

* Only uses a single master cluster. While it can fall over to a new master cluster if the current master cluster fails, this process can take up to a minute.
* MongoDB's single master node limits the ability to write to the database. Since writing new data to the MongoDB database is restricted by the capacity of that master node, the database is prone to limited capacity.

# **Comparison**

The following are the comparison between the three databases: MongoDB, Cassandra, and Hbase.

* **Data availability**: Cassandra supports multiple master clusters while MongoDB and HBase use a single master cluster. If a MongoDB or HBase master cluster fails, delays in large dataset processing and management usually follow.
* **Scalability**. MongoDB and HBase’s single master node limit their write scalability. Cassandra’s multiple master nodes enhance and increase scalability .
* **Data modeling.** MongoDB is document and object-oriented. Cassandra and HBase use traditional tabular column and row structure.
* **Query Language**. MongoDB only supports JSON-like queries. HBase works well with Hive, a query engine for batch processing of big data. Cassandra has its own query language (CQL). Query intensive data management favors Cassandra.
* **NoSQL Schema.** MongoDB does not always require a scheme, allowing for files of different structures to be stored, analyzed and interpreted. Cassandra and HBase are more stationary NoSQLs that are less flexible. [5]



# **Conclusion**

Cassandra vs. MangoDB :Cassandra has the upper hand in terms of its high- availability. The highly distributed architecture allows us to write to a cluster even when nodes fail. Cassandra’s reputation for fast write and read performance, and distributing accurate linear scale performance in a masterless, scale-out design, elites its topmost NoSQL database rivals in many use cases. On the other hand, MongoDB is great for storing unstructured data. It has schema-free architecture which is compatible for quick caching and logging. Real-time analytics and streaming applications count on rapid caching and logging operations. MongoDB is also significant for fast query times since it supports secondary indexes. Cassandra will be a better fit if expecting for data operations to scale rapidly. The overall comparison concludes that Cassandra performs better with heavy data loads as it can support multiple master nodes in a cluster. [6]

Cassandra vs. HBase : Fast writes into the database is a unique feature of Cassandra. HBase, on the other hand, provided write speeds that were nearly twice as fast as the traditional relational database, MySQL. The fact that Cassandra incorporates within itself, simultaneously, the properties of Amazon’s Dynamo and Google’s Big Table makes it really suitable for applications that perform heavy write operations and need to have low write latency. If you are looking for an always-available system, then Cassandra might be a better choice. [7]

## References

1. <https://immagic.com/eLibrary/ARCHIVES/GENERAL/WIKIPEDI/W120911A.pdf>
2. <https://ieeexplore.ieee.org/abstract/document/9838762> → check it
3. Hassan, M.U.; Yaqoob, I.; Zulfiqar, S.; Hameed, I.A. A Comprehensive Study of HBase Storage Architecture—A Systematic Literature Review. Symmetry 2021, 13, 109. https://doi.org/10.3390/ sym13010109
4. <https://www.researchgate.net/publication/340348718_HBase_or_Cassandra_A_Comparative_study_of_NoSQL_Database_Performance/citations>
5. <https://www.researchgate.net/publication/355480192_Comparison_of_Hbase_Cassandra_and_MongoDB>
6. Anusha, K., Rajesh, N., Kavitha, M., & Ravinder, N. (2021). Comparative Study of MongoDB vs Cassandra in big data analytics. 2021 5th International Conference on Computing Methodologies and Communication (ICCMC). doi:10.1109/iccmc51019.2021.9418441
7. ogi, V. D., & Sinha, A. (2016). Performance evaluation of MySQL, Cassandra and HBase for heavy write operation. 2016 3rd International Conference on Recent Advances in Information Technology (RAIT). doi:10.1109/rait.2016.7507964
8. <https://www.tutorialspoint.com/cassandra/cassandra_architecture.htm>
9. <https://logz.io/blog/nosql-database-comparison/#:~:text=Cassandra%20Advantages%20and%20Use%20Cases,be%20used%20to%20achieve%20quorum>.

**SELECTION**

**EXTRACTION:** Apache Kafka

**TRANSFORMATION:** Apache Spark

**LOAD:** Cassandra

Checked By Sir Shuja : 16-11-2022