Big-Scale Data Processing Frameworks Research

Tong He

CONTENT

1. Abstract

Any activity people do in the information era is inextricably entwined with data and it is estimated the size of the 'digital universe' will explode to 170 ZB by 2025, with as much of the data residing in the cloud as in data centers and about 30 percent of the data generated to be consumed in real-time. The initial design of Apache Hadoop was tightly concentrated on processing MapReduce tasks in clusters. In many organizations and web companies, the Hadoop cluster is the common platform in which operational data are stored and batch-processed.

As increasingly diverse companies work for it, Hadoop revolutionized itself though enhancing the limitations of scalability and fault tolerance, programming flexibility and utilizations of cluster resources. Moreover, a new series of execution frameworks have been added to it, including Spark, Flink, Storm and Samza, each with its own strengths and weaknesses. Storm and Samza are proprietary streaming frameworks while Spark and Flink are hybrid solutions. These frameworks complement complex computing capabilities such as machine learning and graph computing. Up to date, Spark and Flink are heavyweights leading from the front in terms of developments. In this report, we introduce the architecture and core modules of the five frameworks and further compares the trade-off, capabilities and application scenarios of these five frameworks. Finally, sharing some pointers as to how to choose the best framework for different use cases.

2. Introduction

Data locations are categorized into three fields. First one is the core, involving physical data centers and cloud, second one is the edge including cell towers and branch offices and lastly, endpoints such as PCs, phones and Internet of Things (loT) devices. As web services and IoT products ushered in the explosive growth and we have at fingertips a combination of global environmental data, people are exploring how to use big data to get a big boost. In the early days of big data, batch processing infrastructure worked as the only way to store and process big data, but now since information shift to mobile, where companies are crying for real-time intelligence to keep up with network demands and functionality, streaming processing becomes pervasive. In this report, we will introduce one of the most essential ingredient of a big data service: processing frameworks. First of all, the report introduces five popular frameworks including Hadoop, Spark, Storm, Samza and Flink. Also, providing a detailed

comparison among them from three aspects including design trade-off, capabilities and application scenarios. Finally, this report discusses the future possibilities regarding the demand for advanced feature.

3.Frameworks

The processing framework and processing engine are responsible for data computing in data systems. Although there is no authoritative definition of the distinction between "engine" and "framework", most of the time the former can be defined as the module that is actually for processing data operations, while the latter can be defined as a series of modules that take on similar roles. For example, Apache Hadoop is typically regarded as a framework that uses MapReduce engine as the default data processor for the engine and framework can typically be used interchangeably or together. Apache Spark, the framework to be discussed later, can be incorporated into Hadoop and replace MapReduce. This interoperability between modules in different frameworks is one of the reasons why big data systems are so flexible.

3.1 Hadoop

To distribute files and computing tasks over clusters is the prime motivator for Apache Hadoop. In 2003, Google developed the “Google File System” to give efficient, reliable, scalable access to data on top of clusters [1]. One year later, they built the “MapReduce” model to enable programmers to easily carry out large scale parallel computations [2]. The core Hadoop modules were inspired by the two papers. After Apache Software foundation hosted Hadoop that derived new modules and related projects over the years, now the term “Hadoop” refers to basic modules or the ecosystem (a family of services under the umbrella of Hadoop infrastructure for complement functionality), or the collection of additional projects built on that foundation, providing complementary services to it.

3.1.1 HDFS

The Hadoop distributed file system (HDFS) is a distributed, scalable, and portable file system written in Java, designed for the throughput of extremely large files (terabytes, Petabytes) with streaming data access patterns, operating on large clusters of commodity machines. Three main components of HDFS:

- NameNode (a.k.a. Master node): HDFS has nominally a single NameNode which does not store actual dataset, but stores Metadata including a number of blocks, their

locations, on which Rack, all the locations of DataNodes storing the corresponding blocks, where the replications are stored other details. It is responsible for namespace hierarchy management, client's access to files regulations and filesystem execution like naming, closing, opening files and directories, through maintaining the filesystem tree and the metadata.Namespace and metadata are stored in RAM but periodically flushed to disk [3]. Modification log keeps the on-disk image up to date. The namespace image and the edit log are stored persistently as two separate files on the local disk.

- DataNode (a.k.a Slave): HDFS DataNodes are the workhorses of HDFS which carry out actual data storing in the local file system, block replica creation, deletion as wells replication, immediate block report sending, and re-register or shutdown on the instructions of NameNode. At the beginning, DataNodes connect to NameNode by handshaking executing verification of namespace ID and software version of DataNode [4]. During running, each DataNode tell the NameNode in Heartbeat mechanism at regular intervals to inform alive; when NameNode no longer receives it, it will regard the DataNode as dead and start to replicate what was stored in dead DataNode to some other one.

- HDFS client: refer to HDFS interface used to interacts with NameNode and DataNode on behalf of user to fulfil user request. Different read or write operations corresponding to different types of client.

3.1.2 Hadoop MapReduce (MRv1)

Hadoop MapReduce is a programming paradigm for easily writing applications that process an enormous amount of structured and unstructured data stored in HDFS. Developer specifies a map function and a reduce function for one task of a job as Map and Reduce phase. Following the programming specifications, programmers just need to write a small amount of job logic code to implement a powerful parallel program for big data processing.

- Client: programmer's work - write MapReduce programs, configure and submit the job.

- JobTracker (Master): the master daemon responsible for client communications, job and resource management, scheduling.

- TaskTracker(slave): service runs on DataNodes, administering them in which Mapper tasks and Reducer Tasks are executed.

- Scheduler: Decide on the order in which tasks are performed. Current three popular schedulers are FIFO Scheduler, Capacity Scheduler, Fair Scheduler.

At startup, client submits a job to the JobTracker who then validates the request and breaks the job into tasks. Next, it communicates s to the NameNode to find where the data are stored on HDFS and assign the best TaskTrackers for the execution of tasks. During the runtime, JobTracker monitors the individual TaskTrackers with heartbeat mechanism and RPC as explained in HDFS part and then sends back the current job status [5]. When completed, Reducer will aggregate values of Mapper output by key and final answers as key-value pairs to HDFS or next MapReduce function.

As clusters and workloads grow, the limitations of MRv1 gradually emerge:

1). The limitations of scalability and fault tolerance resulting from centralized handling of jobs' control flow. JobTracker holds the dual responsibility of scheduling the execution of tasks as well as allocating the cluster resources. Thus, massive MapReduce jobs come potentially with failure of the master and endless scalability concerns for the scheduler. When the number of nodes is greater than 4000, a cascading failure occurs, resulting in deterioration of the complete cluster [6].

2). Poor programming flexibility due to the tight coupling of the sole programming model on top of the cluster manager, forcing abuse of the MapReduce computing paradigm [7].

3). Waste of cluster resources for two points on the TaskTracker side. Firstly, it is too simple to set the number of Map/Reduce tasks as units of resources, but not to consider CPU/memory usage. If two large memory-consuming tasks are dispatched, it is easy to OOM. Secondly, the resource is mandatory to be divided into map task slot and reduce task slot, leading to a waste of resources when only map or reduce tasks in the system.

3.1.3 Yarn

To fundamentally solve the above performance bottlenecks of the version and underpin a sustainable future of Hadoop, MapReduce framework has been completely rewritten since 0.23.0 release as Hadoop 2.0, which is named MapreduceV2(MRv2) or Yarn.

The core of change is to break the two major functions of JobTracker, that is, resource management and job scheduling/monitoring into two separate daemons, that is, JobTracker and TaskTracker are covered by three components: ResourceManager, NodeManager and ApplicationMaster.

3.1.3.1 Yarn components

- Resource Manager (RM): only focus on the management of cluster resources;
- Application Master (AM): typically a one-per-running-application that manages individual running applications.
- Node Manager (NM): launching and managing containers on a node.
- Container: the abstract notion refers to a collection of resources incorporates memory, CPU, disk, network and so forth.

3.1.3.2 MapReduce execution process on YARN [8] shown in Figure 1:

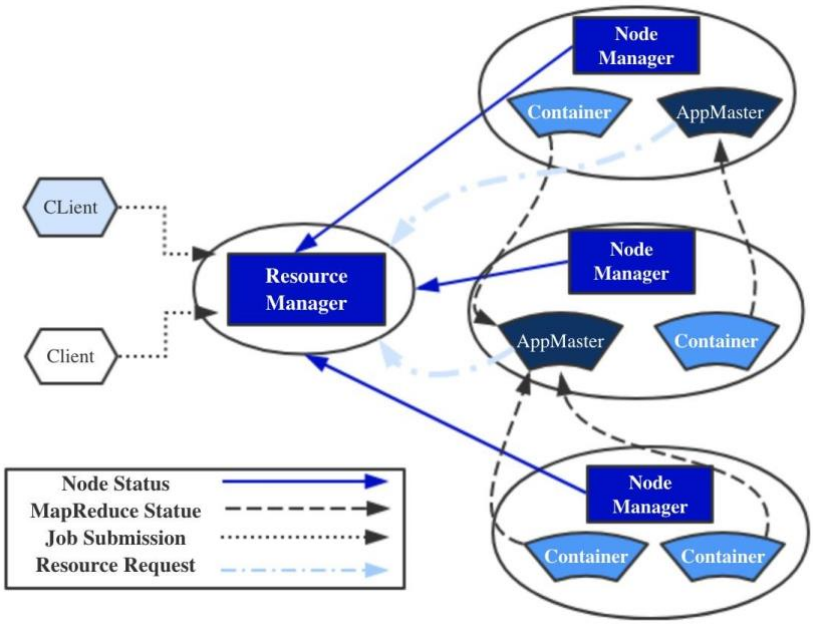

Figure 1. MapReduce execution process on YARN

1) Client submits the job, allocate resources to the RM, and copy the jar package to 10 by default to HDFS

2) RM specifies an NM to open a Container and run an AM in the Container to manage the application.

3) AM calculates the resources required for this application and requests resources from the RM.

4) RM allocates resources, open different containers on NM, and run map task or reduce task in the containers.

5) When all the tasks have been executed, AM will feed back the results to the client. After all the work completed, AM closes itself.

6) If a Map or Reduce task fails, AM will reapply for a new container to execute the task.

3.1.3.3 Yarn (MRv2) over MRv1

With YARN coming to the scene as a part of the Hadoop 2.0 platform, three limitations of Hadoop 1.0 mentioned above are perfectly countered as below:

1). Enhanced fault tolerance and Scalability [7]:

- Each component in Yarn has fault-tolerance mechanism:

  RM automatically restore users' pipelines when fails. Since RM still has a single point of failure as JobTrack in MR1v, it recovers from its own failures by restoring its state from a persistent store on initialization and once complete, kills all the active containers including AM and then launches new instances of each AM.

- When AM fails, RM is responsible for restarting it. As RM stores running tasks, it is no need to rerun the tasks in AM.

- If a task fails in NM, the task JVM will reports the failure to its parent AM who will decide how to deal with the failed task.

As explained above, responsibility is now distributed between the RM and AM by removing overburdened JobTracker, so the master node is freed from the job management, and these resources could be used for management of more node, which is the key principle of improving scalability behind Yarn,

2). Remarkable programming flexibility

YARN has significant performance enhancements resulting from splitting the functionalities [9]. The complete separation of cluster resource management and data processing capabilities allows different purpose-built data processing engines not conforming to MapReduce including, interactive processing, graph processing and streaming as well as batch processing. Also, it allows diverse applications running on Hadoop, such as MPI, Giraph, Spark, Storm, Hoya, and REEF.

3). Optimized utilization of resources

YARN does no longer exist any fixed MapReduce slots for tasks. With YARN as the resource master, relative jobs can share a common resource, thus leading to an increase in efficiency and possibility for graph processing and iterative modelling.

3.1.4 Other modules in Hadoop ecosystem

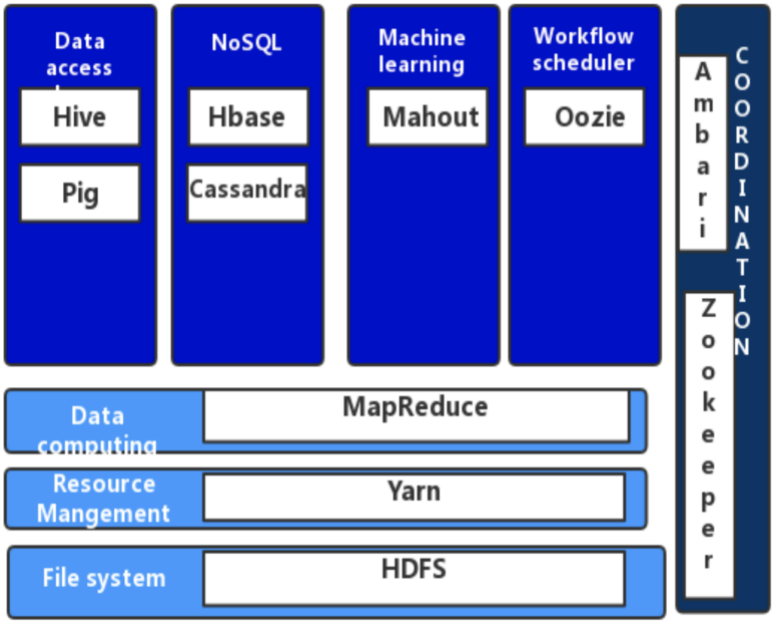

Figure 2. Hadoop 2.0 ecosystem

We have detailed the three core Hadoop modules and the other related projects shown in Figure 2.

Hive: a data warehouse infrastructure that has four main functions: data summarization easy ETL processing, ad hoc querying and analysis.

Hbase: a scalable, distributed, column-friendly oriented NoSQL database that runs on top of HDFS.

Cassandra: a distributed, highly scalable multi-master database and can be used to manage large amounts of structured data. It enables high availability with no single points of failure.

Mahout: library for data mining as well as machine learning.

Pig: a high-level data-flow language and execution framework for parallel computation, geared towards analyzing batch data and coding faster.

Zookeeper: a distributed coordination service to manage a large set of hosts in a distributed scenario. ZooKeeper simplifies the complicated management process with its simple architecture and API, enabling developers to figure out core application logic ignoring the distributed nature.

3.2 Spark

While Hadoop has revolutionized big data processing, rendering all store and process big data, it is geared towards performing simple computing tasks on big datasets. As advanced functionalities are more and more in demand, there is for storing and computing in memory and in having finer-grained control over computing processing. Apache Spark was designed to meet urgent needs, as a powerful unified analytics engine for large-scale distributed data processing ranging from Machine Learning, GraghX, Spark SQL (data warehouse) to Spark Streaming.

3.2.1 Spark stack

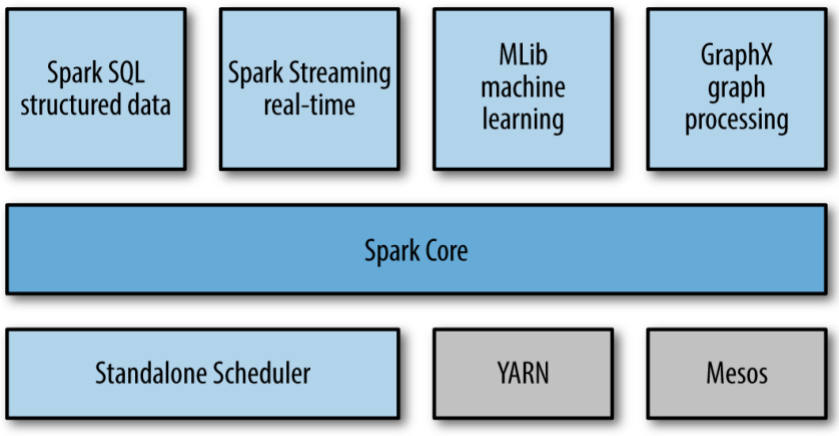

Figure 3. Spark stack

Spark core: the most important module of Spark that supports basic functionalities including task scheduling, memory management, fault recovery and communication with databases.

RDD (Resilient Distributed Dataset): the Spark core data structure, essentially an abstract, distributed object collection which can be Python, Java, or Scala objects, enable manipulating on the different nodes of the cluster and to avoid intermediate result storage. RDD provides two types of functions: Transformations and Actions; Transformation is responsible for transforming the existing RDD into another RDD at intermediate steps. Action works with the actual dataset to return non-RDD data as a final value to the target output system. Only 'Action' actually computes, whereas 'Transformation' just records the dependencies between each other. [10].

Spark SQL: the module supporting SQL or HQL processes structured data from the source including Hive tables, Parquet, and JSON[10]. It provides a programming abstraction, DataFrame, a distributed dataset based on RDD acting as a distributed SQL query engine to be built from structured data files, tables in hive, external relational databases, and RDDs. As RDDs can supports multiple languages, programmers can switch back and forth between APIs to manipulate tasks.

Spark Streaming: the core Spark module supporting scalable, high-throughput, fault-tolerant streaming processing to real-time data [10]. It receives data stream from multiple sources such as Kafka, Flume, Kinesis, or some social media applications such as Facebook and Insgram, and then split them into batches of data at a certain time interval. Spark engine takes out the batch data, encapsulating them into RDDs and then process them as a typical producer-consumer model and after that, results flow into HDFS, databases, and live dashboards.

Mlib: Spark's machine learning (ML) library provides multiple types of machine learning functions as well as lower-level ML primitives. To be more specific, Mlib provides four types of tools respectively are most machine learning functions, storing function works for storing and load modules, Pipeline tools and featurization [11].

GraphX: the library for graphs and graph-paralell computations. GraghX extends the Spark RDD API by defining a new Graph abstraction: a directed multigraph with properties

attached to each vertex and edge [10]. GraphX provides various operators like subgraph, mapVerticesand aggregateMessages, and Pregel API's optimized variant. Besides, it collects updating graph algorithms like PageRank and Triangle counting.

Cluster managers supported in Spark: offer various options for deploying applications and here listed three of them:

- Standalone mode: completely distributed mode with Spark own resource manager. Spark leverages its own Master daemon to manage and monitor resources, which makes Spark cluster more lightweight.

- Spark on Mesos: Mesos is another popular cluster manager can run both analytics workloads and long-running services. Mesos has two mode in which Spark works:

    - Coarse-grained Mode: the running environment of each application consists of a Driver and several Executors. Each Executor occupies several resources, and multiple tasks can be run internally (corresponding to the number of slots). All the resources are applied before tasks officially run, and these resources must be occupied during the running process even if they are actually not used. The advantage is low startup overhead whereas a waste of resources.

    - Fine-grained mode: consider massive waste of resource in Coarse-grained mode, the other mode, fine-grained is based on the idea of on-demand allocation. It allows applications gets more or fewer cores as it ramps up and down, that is, dynamically allocate resources to each executor and the corresponding resources can be released immediately after a single task done. Each task will report status to Mesos slave and master for more fine-grained management and fault tolerance. It is easy to control and isolate resources, whereas the shortcoming is obvious that short operation delay is large with an additional overhead in reporting.[10] [12].

- Spark on Yarn: Spark acts as a client submitting tasks to Yarn, which is responsible for resource management and job scheduling as introduced above. In this mode, Spark is regarded as a Hadoop module under control of Yarn to share system resources. Yarn currently only support coarse-grained model [12].

3.2.2 Four features of Spark:

- Fast Processing: Spark is 100 times faster than Hadoop in memory computing through putting processing data in memory.

- Unified tools: Spark offers a range of higher-level libraries for various application needs, such as SQL queries, data streaming processing, ML and graph computing. These comprehensive APIs simplify programming works and enhance the productivity.

- Ease of use: Spark provides high-level APIs which transparent the cluster itself, only focusing on the logic of task implementation, APIs mainly support Scala, Java, Python, and R, making it easy for users to develop Spark programs. Also, it comes with a built-in set of over 80 high-level operators used interactively to query data within the shell [13].

- Compatible with other infrastructure: Spark is perfectly compatible with all the Hadoop family as the Hadoop computing engine. It provides good API to connect with other software such as cluster managers, databases, publish-subscribe messaging systems and so on.

3.3 Storm

Although the frameworks Hadoop and Spark are extensively applied in big data analysis field, they both have bad performance on real-time process problem. System needs to maintain lots of message queues and supervisors which leads to a complex graph structure. System needs to pay a lot attention to sender, receiver and message serialization. All these make the system vulnerable and not fault-tolerant. That is the reason why Storm is established.

Nathan proposed the core concept of Storm in December, 2012. It is a real-time distributed process system which can process massive data in a very short time.

3.3.1 cluster structure

The Storm cluster adopts the master-slave architecture. The master node is, the slave node is Supervisor, and the scheduling-related information is stored in the Zookeeper cluster [14]. The architecture is shown in the following figure 4. Both of Nimbus and Supervisor have no

state. Job states and heart-beat information are stored in Zookeeper while committed programming resources are in disks.

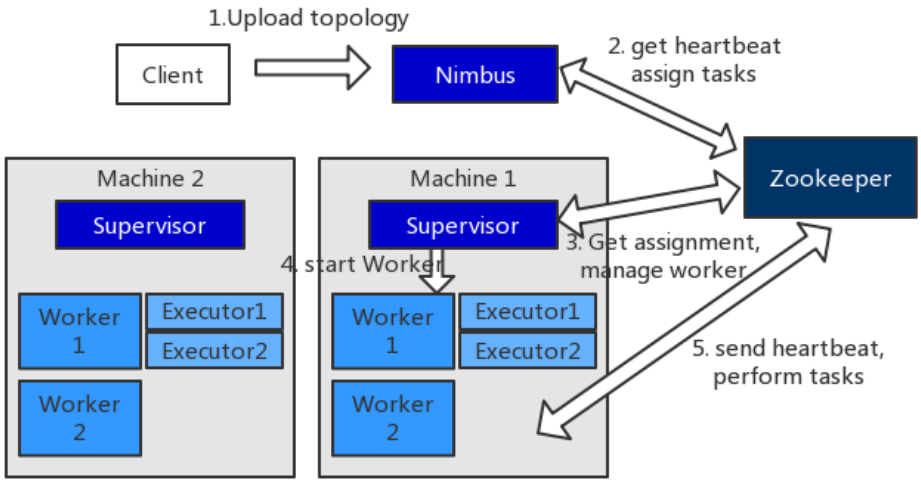

Figure 4. Cluster structure of Storm

- Nimbus: Just like the JobTracker in Hadoop, it is the master node in the cluster and only one in each cluster. There are four main function of Nimbus:

  1.Upload the topology.

  2. Distributing user code across the cluster.

  3. Allocating resource and assignment to the specific Supervisor.

  4. Monitor the state of the whole cluster like whether the worker is working.

- Supervisor: It is responsible for monitoring the work assigned to it and start/close the worker process as needed. It must be deployed on each machine that running the storm.

- Zookeeper: The external resource that Storm focus on. It stores the heartbeats of Nimbus, Supervisor and Worker. Nimbus also performs scheduling and task assignment based on the heartbeat and task performance on the Zookeeper.

3.3.2 Data process

Storm implements a model of data flow that includes the following core concept.

Topology is the program that the storm commits to run is called a topology which just like an application. It packages the logic of a real-time handler. Actually, it is a complex multi-stage flow calculation similar to MapReduce. The main difference is that the MapReduce task ends up after finishing tasks while the topology always run. A topology is a topology that connects Spout and Bolt together by stream grouping. Each edge of the graph represents that a Bolt subscribes to output streams of other Bolt or Spout [15].

Input data source: There are many sources of data and Storm allows users to set simple data by themselves like HDFS, Kafka and Mongodb.

Tuple is a list of ordered elements which is the main data structure in Storm. It can be used to wrap the data you need to actually process. It is the basic unit of a message composed of immutable Key-Value pairs. It supports all data types in the default situation.

Stream is stream consists of an infinite sequence of tuples.

Spout converts data received from data source into Tuple and spits it into a topology.

Bolt is the real logical processing unit with an execute method internally, which can be inherited and implemented by the user to implement its own processing logic. A Bolt can handle any number of input streams, producing any number of new output streams. It can also do function processing, merging, aggregation and store stream to the database and other operations.

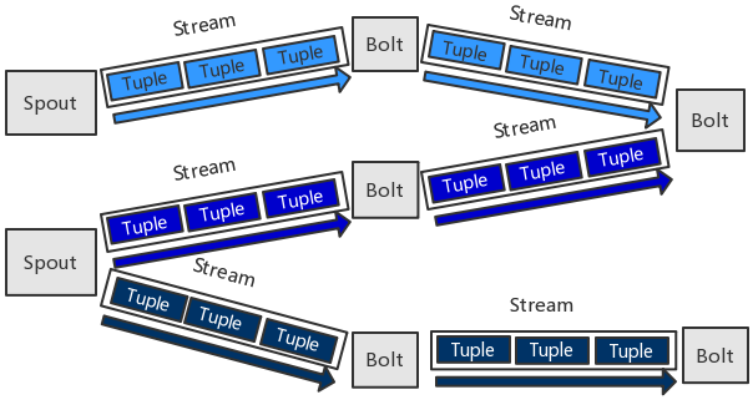

Figure 5. Data flow model of Storm

The figure 5 shows the complete process flow. Client submit topology to Nimbus. Nimbus can communicate with Supervisor through ZooKeeper. Nimbus send assignment allocation to

Zookeeper who will then pass it to corresponding Supervisor. After receiving tasks, Supervisor opens Worker and monitor the state of Worker. One Worker is a process that can start several Executor to execute tasks. Nimbus, Supervisor and Executor always upload heartbeat information to ZooKeeper and Nimbus can monitor the state of cluster by checking heartbeat package in the ZooKeeper.

The most important abstraction in Storm should be Stream grouping. It can control how the tasks corresponding to Spot / Bolt distribute Tuples and launch Tuples to the tasks corresponding to Spot/Bolt. Strom offers eight built-in implementations, the most commonly used are Shuffle Grouping, Fields Grouping, and Direct Grouping [15].

3.4 Samza

Samza is a technology open sourced by LinkedIn which is also a distributed stream processing framework like Storm. The structure of Samza is similar to Hadoop but uses LinkedIn's own Kafka distributed messaging system and the resource manager Apache Hadoop YARN [16].

Samza consists of three layers and provide support for all three layers:

Streaming media: Kafka

Execution: YARN

Processing: Samza API

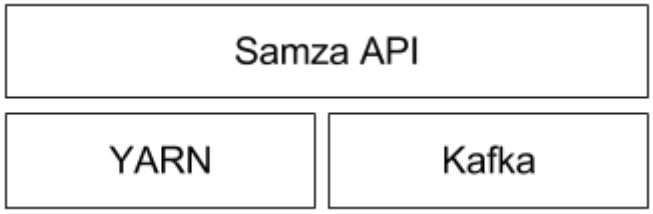

Figure 6. Structure of Samza

3.4.1 Streaming media

Kafka is a distributed message queuing system that provides at least one message delivery guarantee and highly available partitions. The system guarantees that no messages are lost, but in some cases, the consumer may receive the same message multiple times.

Each stream in Kafka is called as a topic and each topic is split and copied to multiple machines called brokers. The partition of the stream is still available even if the machine is powered down. A producer sends a message with a key that determines which partition the message should be sent to. The Kafka agent receives and stores messages sent by the producer. Kafka users can then read themes by subscribing to messages on all partitions of the theme.

3.4.2 Execution layer

The structure of Yarn has been explained in section 3.1.3. It will not be repeated here. In short, it consists three part: ResourceManager, NodeManager and ApplicationManager. The below figure 7 shows the analysis process [17].

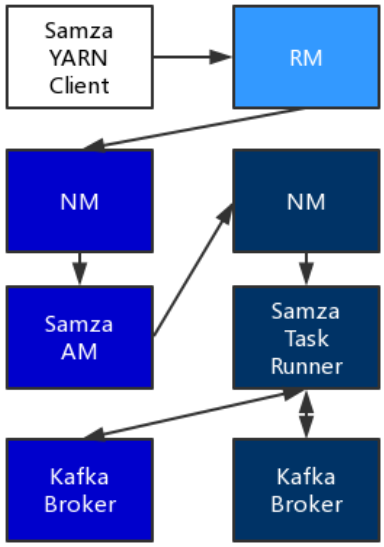

Figure 7. Execution process.

1. When the Samza client needs to execute a Samza job, it will submit a job request to YARN's ResouceManager.

2.ResouceManager runs Samza ApplicationMaster by communicating with the NodeManager to allocate a container (including CPU, memory, etc.) for the job.

3.Samza ApplicationMaster further requests the ResourceManager to run the container for the task.

4. After obtaining the container, Samza ApplicationMaster communicates with the NodeManager where the container is located, launches the container, and runs Samza Task Runner in it.

5.Samza Task Runner is responsible for executing specific Samza tasks and completing stream data processing analysis.

3.4.3 Data processing

Samza handles the stream that consists of an immutable message of a similar type [16].

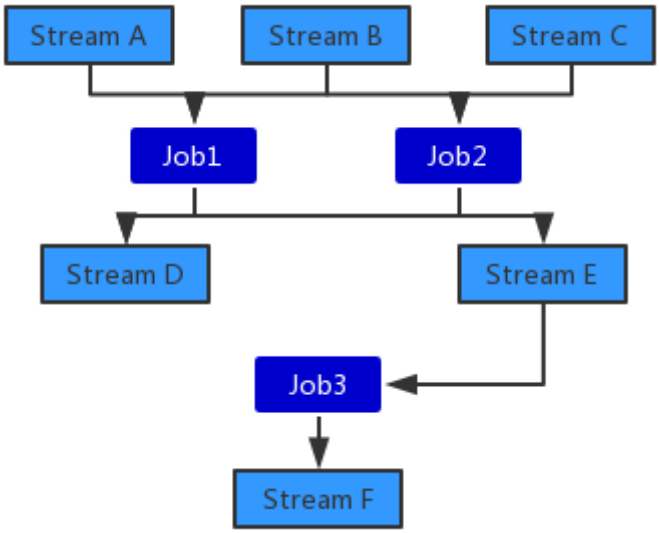

Figure 8. structure of data processing.

To enhance scalability, each stream is decomposed into one or more partitions. Each partition in the stream is a fully ordered sequence of messages. Each message in the sequence has a specific identifier called an offset. A message only needs to attach to a partition of the topic when it is added to a stream [18].

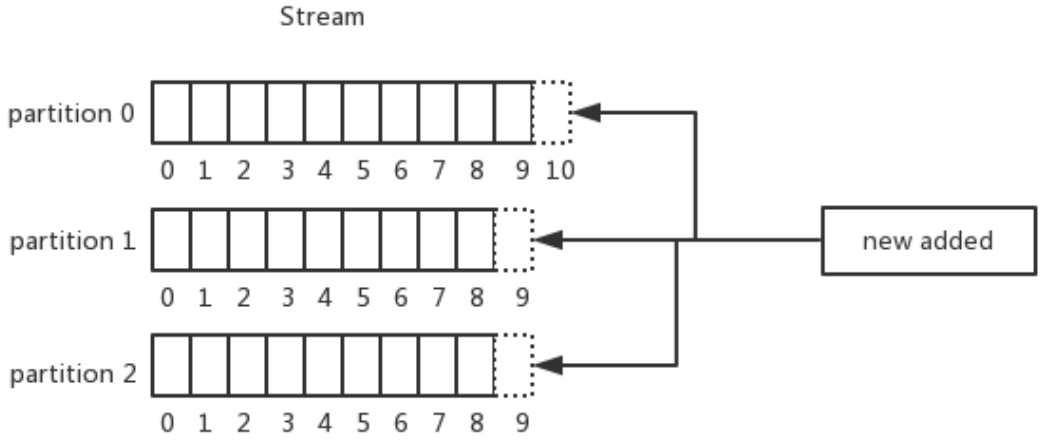

Figure 9. partition stream.

A Samza job is code blocks that execute a logical transformation on input streams to get output messages. Samza scales the job by breaking it down into multiple tasks. Each task processes data from one partition of input stream of each job. Tasks sequentially handle messages of partition according to offsets of messages. Partitions have no definite order relationship so that each task can run independently. Then, YARN allocates tasks to machines which makes the entire job can be completed in a distributed way.

Multiple jobs can be combined to create a data flow graph whose edge is the stream containing the data and node is the job that performs the conversion. This combination is done entirely through the job as a stream of inputs and outputs. Jobs are completely decoupled which means they do not need to be implemented by using same language and the upstream jobs will not be affected by the change of downstream jobs.

3.5 Flink

Flink is a hybrid framework like Spark mentioned above. Compared with Spark, Flink has a different processing logic. Spark divides streaming data into several small batches, while in Flink, all data is treated as a stream, batching data is regarded as streaming data with a boundary, one data item is processed at a time.

3.5.1 Basic architecture

Flink is a hybrid framework like Spark mentioned above. Compared with Spark, Flink has a different processing logic. Spark divides streaming data into several small batches, while in

Flink, all data is treated as a stream, batching data is regarded as streaming data with a boundary, one data item is processed at a time.

The architecture of Flink system is also similar to Spark, a master-slave style architecture [19] (shown in Figure 10). There are three main parts in Flink cluster:

- JobManager

  JobManager is the coordinator of the Flink system. It is mainly responsible for accepting jobs from Client and collecting job status information from TaskManager. Meanwhile, it can schedule the execution of multiple tasks and manage TaskManager.

- TaskManager

  The TaskManager is a worker for actually performing the data computation. TaskManager sets slots at startup [19]. Each Slot can start a Task with a thread. Each TaskManager is responsible for managing resource information on its node, such as memory, disk, network, and reporting the status of resources to JobManager at startup. TaskManager has two phases: registration phase and operational phase.

- Client

  Client is responsible for submitting Job and can be running on any machine (connected to the JobManager environment). The Client needs to preprocess the user-submitted Flink program before submitting it to the Flink cluster for processing. The Client can establish a connection to the JobManager by obtaining the address from Flink program configuration, and then submit the Flink Job to the JobManager.

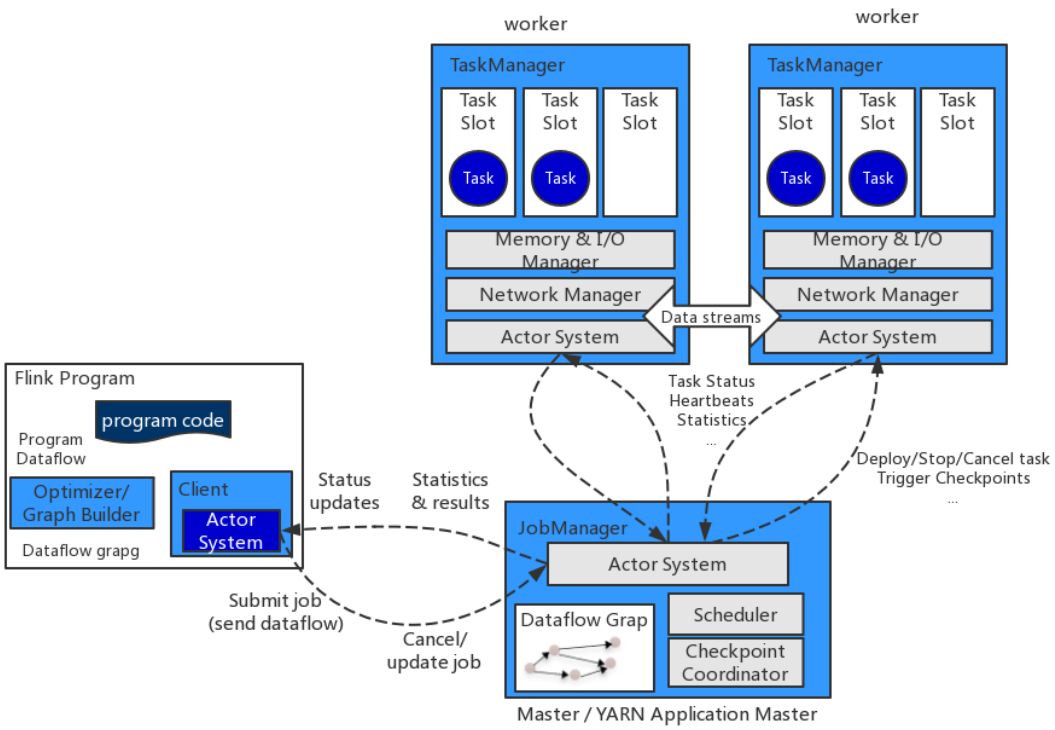

Figure 10. The Flink System Architecture

When a Flink cluster is started, the JobManger and TaskManagers will be run at first. Once the Client submits tasks to JobManager, JobManager dispatches tasks to each TaskManager for executing, then TaskManager needs to report the heartbeat and statistics to JobManager. The communication between JobManager and TaskManager follows the RPC-based asynchronous communication protocol. It can be seen that task scheduling in Flink is a multi-thread model, and different tasks are mixed in one TaskManager process.

3.5.2 Software Stack

Flink is a layered architecture system. The components contained in each layer provide specific abstractions to serve the upper components [20].

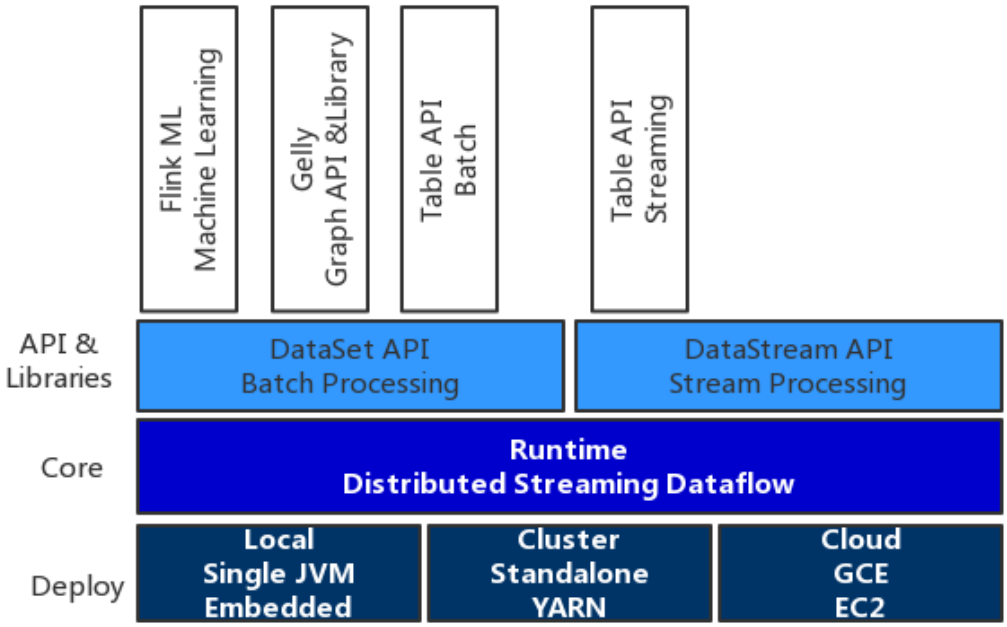

Figure 11. The Flink Software Stack

- Deployment layer

  This layer focuses on the deployment patterns of Flink, which supports several deployment patterns: local, cluster (Standalone/YARN) and cloud (GCE/EC2). Standalone deployment mode is similar to Spark.

- Runtime layer

  This layer provides all core implementations which support Flink data computing, it also provides basic services for the upper API layer, for instance, supporting for distributed Stream processing, mapping from JobGraph to ExecutionGraph, scheduling, etc.

- API layer

  This layer mainly implements the Stream processing API oriented to unbounded Stream (DataStream API) and Batch processing API (DataSet API).

- Libraries layer

This layer is built on top of API layer to meet the needs of specific applications. According to the division of API layer, it also corresponds to stream-oriented processing and batch-oriented processing respectively. Stream-oriented processing support: CEP (complex event processing) and SQL-based operations (table-based relational operations), while Batch-oriented processing support FlinkML (machine learning library) and Gelly (graph processing).

3.5.3 Architecture based on Yarn level

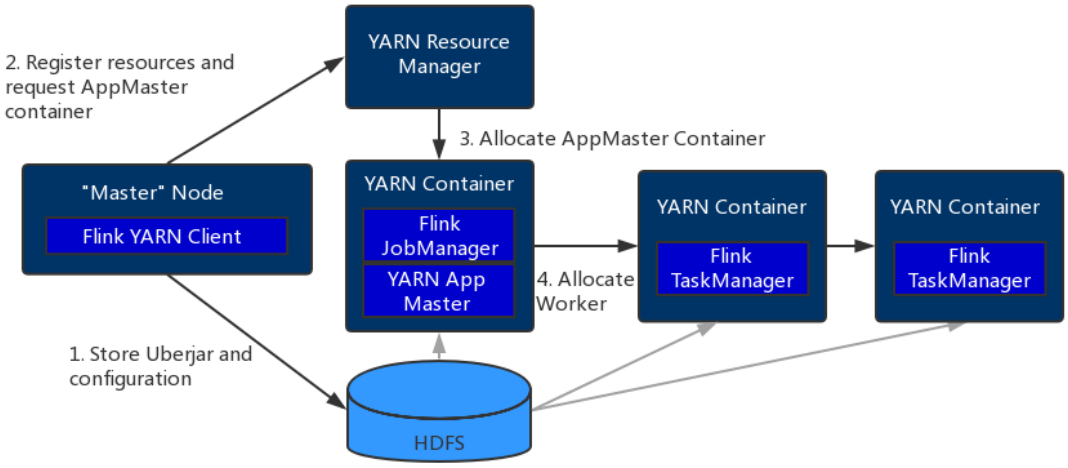

Figure 12. Yarn-based architecture of Flink

The yarn-based architecture is similar to spark, in which the Client submits the App to the Resource Manager (RM) to run, and then RM assigns the first container to run AppMaster(AM), and AM is responsible for the supervision and management of resources [21]. It should be noted that the yarn mode of Flink is more similar to the cluster mode of spark on yarn. JobManager will be started in a container to do task scheduling and distributing. Yarn AM is in the same Container with JobManager, hence AM can know the address of JobManager and then apply for Container to run TaskManager. After Flink successfully runs on the YARN cluster, the Flink YARN Client can submit the Flink Job to JobManager for subsequent mapping, scheduling and computing.

3.5.4 Characteristics

- High throughput & low latency.

- Support the window mechanism of Event Time semantics [22].
- Lightweight mechanism for fault tolerance. High throughput and strong consistency can be achieved.
- Memory management can be done in the JVM. Applications can exceed the limit of the main memory and bear less overhead of garbage collection.
- Flink stack provides a lot of advanced API and class library to meet different scenarios: machine learning, graph analysis, relational data processing.
- Extensive integration. Flink integrates with many projects in the open source big data processing ecosystem. Flink can run on YARN, work in collaboration with HDFS, read stream data from Kafka, execute Hadoop program code, and connect multiple data storage systems.
- Flink can be deployed separately from Hadoop. Deployment only depends on Java environment, which is relatively simple

4 comparison and analysis

After being introduced separately, these five frameworks will be compared from different aspect in this section

4.1 Comparison of the approaches based on design.

In fact, there are two modes for big data processing: batch processing and stream processing. Meanwhile, frameworks are divided into three categories which are batch-only processing, stream only processing and hybrid framework. The hybrid framework can achieve both batch and stream processing.

In this section, firstly, I will summarize the differences between the two processing modes and then compare the three different frameworks.

4.1.1 Difference between batch processing and stream processing.

Batch processing consists of breaking down the task into smaller tasks, performing calculations on each machine in the cluster, recombining the data based on the intermediate results, and finally calculating to get the final results which will be rewritten to disks. Batch

processing systems are most suited for very large data sets. Stream processing, on the other hand, operates on a stream of data consisting of continuous single data items, focusing on the timeliness of data processing results.

At the execution engine level, the biggest difference between a batch and a stream processing system is the way data is transmitted between nodes. For a streaming processing system, the standard model for data transmission between nodes is: when a piece of data is processed, it is serialized into the cache, and then immediately transmitted to the next node through the network, and continues processing by the next node [23]. For a batch system, the standard model for data transfer between nodes is: when a piece of data is processed, it is serialized into the cache and is not immediately transmitted to the next node through the network. When the cache is full, it lasts into local disk. Then all the data is processed, the processed data is transmitted to the next node through the network.

4.1.2 Differences between the three types of frameworks.

In general, hybrid framework can handle both batch and streaming data. But the design of Flink and Spark are totally different. Flink can support batch and streaming process by a single engine just with different parameter. However, Spark streaming, as an extension of Spark core, it provides stream processing ability which is not achieved in Spark core. Actually, Flink is the only one that can use a single engine to handle both two data type. As a result, In the next comparison in this section, Flink will be the only hybrid framework while Spark core and Spark streaming will be classified as batch processing and stream processing separately.

4.1.2.1 Batch processing framework

Hadoop is the first full framework which uses MapReduce engine to do batch processing. The workflow of MapReduce has been introduced before in section 3.1.2. According to that, it can be concluded that Hadoop need multiple disk reads and writes when processing a task since all intermediate calculated result must be stored. Totally different from MapReduce, the process of Spark core's data processing is all done in memory, and it only needs to read once from disks to load data into memory and write once to persists the final result [24]. And since it uses memory to store all intermediate processing results, the process speed of spark is much higher than Hadoop.

4.1.2.2 stream processing framework

There are three stream processing frameworks: Spark streaming, Storm and Samza.

Spark Streaming receives real-time streaming data, batches the data in batches at regular intervals which converts streaming data into batches of time-slot data, and encapsulates each batch of data into an RDD. The group RDD constitutes a Dstream[11]. In general, if the stream data is divided into batches, it passes through a FIFO queue, and then the Spark Engine sequentially takes out batch data from the queue, encapsulates the batch data into an RDD, and then processes it. It is a typical producer consumer model.

Storm's stream processing can orchestrate the DAG (Directed Acyclic Graph) named Topology in the framework. These topologies record the different operation that need to be performed on each Tuple. Topology includes stream, spout and bolt.

Samza relies on Kafka's semantically defined stream handling. Samza's goal is to treat the stream as an accepted message. At the same time, Samza's stream initial element is a message and the stream is divided into partitions. Each partition is a sort of read-only message.

This table summarizes their data processing model.

	Spark	Storm	Samza
Stream Source	receivers	Spouts	Consumers
Stream Primitive	Dstream	Tuple	Message
Stream computation	Transformations Window Operation	Bolts	tasks

Table 1. comparison of stream process.

4.1.2.3 hybrid framework

In Apache Flink, batch data are treated as streams whose boundary is finite which means it can be processed as a subset of stream

Flink performs network data transmission in units of fixed cache blocks. The user can specify the transmission timing of the cache block by the cache block timeout value. If the timeout value of the cache block is 0, the data transfer mode of Flink is similar to the standard model of the stream processing system mentioned above, and the system can obtain the lowest processing delay. If the timeout value of the cache block is infinite, the data transfer mode of Flink is similar to the standard model of the batch system mentioned above, and the system can obtain the highest throughput. At the same time, the timeout value of the cache block can also be set to any value between 0 and infinity. The smaller the timeout threshold of the cache block, the lower the data processing delay of the Flink stream processing execution engine, but the throughput will also decrease, and vice versa. By adjusting the timeout threshold of the cache block, users can flexibly weigh system latency and throughput as needed.

4.2 comparison on capabilities

This section compares the performances and capabilities of the five common big data frameworks. According to the data processing approach and the timeliness of the results, frameworks can be divided into three categories: batch-only framework, stream-only framework and Hybrid framework.

4.2.1 Compare characteristics of three type frameworks

4.2.1.1. Batch-only framework

In batch-only systems, there are large amounts of static data. It keeps waiting until all processing is completed, then returns results. Datasets in batch-only frameworks generally have three features:

- Bounded. The data in the data set must be finite.
- Persistent. The data storage and processing rely on persistent system, for instance, hard disk and database.
- Large. Batch processing can efficiently process large historical data.

Batch systems are usually used to deal with large amounts of historical data due to their excellent performance in dealing with big-scale data. However, the batch-only framework is generally unsuitable for situations with specific requirements on delay because

processing massive data needs a lot of time. Apache Hadoop is the most well-known batch framework.

4.2.1.2. Stream-only framework

The stream-only system processes data as it is loaded into the system in real-time. This is a very different approach than the batch-only mode. The difference is that the stream-only framework does not support to deal with historical data, it can only process real-time data entered from the client. Dataset in this framework is infinite, processing results are always changing dynamically as data-updating.

Stream-only systems has two different data processing types [25], the first type is item-by-item processing which processes unit data at one time, this is a truly real-time approach. Another type is micro-batch, it treats the data entered in a short period of time as a micro-batch and processes these data at a time. No matter which kind of processing approach, its timeliness is more efficient than batch-only framework. Hence, this kind of framework is more appropriate for those applications with high real-time requirements, for example, dynamic analysis of logs and website traffic. Apache Storm and Apache Samza are two typical stream processing systems.

4.2.1.3. Hybrid Framework

This kind of framework can deal with both batch and stream data. It can work with both historical and real-time data by using same or related APIs. Currently, the mainstream mixed processing frameworks are Spark and Flink. Although using only one data processing method may be well suited to a specific environment, the hybrid data processing framework is more compatible since both stream and batch data can be processed. This framework not only provides the methods needed to process data, but also provides its own integration items, libraries and tools, which can be competent for graphic analysis, machine learning, interactive query and other tasks.

4.2.2 Compare Hadoop, Spark, Storm, Samza and Flink

4.2.2.1. Hadoop

Although Hadoop is used to handle complex data, it is actually quite simple. If your data can be processed in batches, broken up into small processing tasks, distributed to a computing cluster, then the data is probably well suited for Hadoop processing.

But this approach relies heavily on persistent storage, hence it is relatively slow and has multiple read and write operations per task. Because servers usually provide a large amount of hard disk storage space, MapReduce can deal with a large amount of data. Compared with other memory-dependent frameworks, it does not require much memory, therefore the cost is not high [26]. A complete Hadoop cluster can be achieved with less expensive components, this cheap and efficient big data framework can be flexibly applied in many cases. The HDFS and YARN resource manager can be combined with other data-processing technologies.

4.2.2.2. Spark

Spark adopts more advanced architecture, which makes it more advantageous than Hadoop in terms of flexibility, ease of use and performance. Spark is faster than MapReduce when dealing with data due to its memory management approaches and advanced DAG scheduling mechanisms.

Another important advantage of Spark is its diversity. This framework has two deployment methods, on own standalone cluster or on Hadoop cluster. Spark can handle both batch and stream data and deal with various jobs.

In addition to the capabilities of the engine itself, an ecosystem of libraries has been built around Spark to provide better support for tasks such as machine learning and interactive queries.

Because memory is typically more expensive than persistent storage, Spark needs to costs more on memory than Hadoop framework. Nevertheless, Spark framework with high processing speed can complete jobs more quickly. Hence for situations where throughput rate is more important than delay, Spark Streaming is more suitable to use.

4.2.2.3. Storm

Storm is designed to handle unlimited streams easily and can be used in any programming language. Each node processes millions of tuples per second and is highly scalable. It can be used for real-time analysis, distributed machine learning, and a large number of other

situations, especially large data streams. Storm can run on top of YARN and integrate into the Hadoop ecosystem. Five features of Storm are fast, scalable, fault tolerance, reliable and easy to operate. However, Batch processing is not supported.

Storm is an ideal choice if one project only needs to process stream data and require a low latency. Continuous data entering the framework will be processed in real time. If batch data needs to be processed, it needs to be solved in combination with other frameworks. In addition, it is friendly and supports the development of many programming languages.

4.2.2.4. Samza

Samza's reliance on kafka-like query systems seems to be a limitation, but it also provides some unique guarantees and functionality that other stream processing systems don't have.

Apache Samza is a pure stream processing framework which can be tightly bound to the Apache Kafka messaging system. Although Kafka can process stream data itself, Samza can make better use of Kafka's architectural advantages. For example, Kafka can provide fault tolerance, buffering, and state storage functions. In addition, Samza can use YARN as a resource manager, Hadoop clusters are necessary if YARN is used [27].

If Hadoop and Kafka have been deployed and stream data needs to be processed in real-time, samza can be considered.Using Samza technology to process stream data makes data-processing job easier, meanwhile it can meet the high requirement of delay. For an organization with multiple groups and stream data, Samza may be the best choice if each group executes streaming process at different stages.

4.2.2.5. Flink

Flink is real stream-oriented framework while Spark is batch-oriented. If one project deals with real time data, Flink is more suitable. It can achieve low latency, high throughput, almost item by item processing.

Many of Flink's components are self-managed. This is a rare practice, but for performance reasons, the technology manages memory without relying on native Java garbage collection.

As for user tools, Flink provides a web-based scheduling view that makes it easy to schedule tasks and view system state. If the client wants to know how the task it sent is processed in the cluster, it can get it by checking the optimization results returned by the cluster.

Flink has great extensibility and compatibility. This technology can easily be integrated with YARN, HDFS, and Kafka. Meanwhile, by using some compatible packages, Flink can correctly run jobs of other large-scale data processing systems, for instance, Hadoop and Storm.

Table 2 shows a comparison of several characteristics of the five frameworks [28].

Features		Hadoop	Spark	Storm	Samza	Flink
1	Scalability	incredible scalability	scalable (The number of nodes in the cluster can be increased)	scalable (The number of nodes in the cluster can be increased)	scalable (The number of nodes in the cluster can be increased)	scalable (The number of nodes in the cluster can be increased)
2	Message delivery guarantee in case of failure	Exactly-once	Exactly-once	At-least once	At-least once	Exactly-once
3	Streaming/batching	only batching	streaming data as micro batching	only streaming	only streaming	batching data as bordered streaming
4	Cost	Less cost on RAM due to MapReduce relies on persistent storage	more cost on RAM	more cost on RAM	more cost on RAM	more cost on RAM
5	Data computation	Disk-based hence full recovery is easy	In memory. data is more prone to loss	In memory. data is more prone to loss	In memory. data is more prone to loss	In memory. data is more prone to loss
6	Auto-scaling	Yes	Yes	No	No	No
7	Languages supported	Primarily Java, but other languages like C, C++, Ruby, Groovy, Perl, Python also supported using Hadoop streaming	Java, Scala, python and R	Python, Ruby, and any JVM-based language	Scala, Java, and Python	Java as well as Scala

Table 2. Comparison of different frameworks

4.3 Discussion of choosing the approaches in different situations

Hadoop ecosystem is the cornerstone of all other frameworks as to the classic batch processing. It has been widely applied to help companies run offline analysis on their product or market tendency based on massive data. For instance, financial institutions apply corresponding applications to assess risk, developing neural network tools, build investment

models and etc. Retailers through this analyse customer behaviours to get a precise and architectural profile of their customers, thus developing their recommendation systems.

Storm does real-time analysis of massive logs which cannot get directly count statistics, such as API calling status of the gateway system generated form the application systems. These logs need preprocessing before streaming to databases.2. Storm perfectly fits when a low-latency response, say analysis in milliseconds is required in a business scenario. A typical case is website performance monitoring, such as Ctrip, which uses the performance standards provided by HTML5 to get the available metrics and log and the Storm cluster analyzes logs and warehousing in real time. It uses DRPC to aggregate into reports and use historical data comparison and other judgment rules to trigger early warning events [29]. Another common use case is online game version analysis system, where developers and product managers can get continuous reports of the game update and monitor and corresponding analysis results just a few seconds after release, and then immediately balance the game by adjusting parameters. This will greatly shorten the game iteration cycle and enhance the vitality of the game.

Samza is leveraged for other use cases such as data-enrichment, re-partitioning of event streams and computing real-time metrics etc. For instance, Redfin real-time notifications: multi-stage stream processing pipeline starting from identifying candidates, analysing profiles, grouping results, join additional data sources leveraging Samza's support for local state and finally sending data to delivery system. eBay also chose Samza to build PreCog, their horizontally scalable anomaly detection system for Samza perfectly fits the bill (high-performance, fault-tolerant and low-latency) [30].

Spark and Flink both fit (near) real-time services: marketing campaigns, machine learning, loT sensors, log and security monitoring and social media sites. As far as streaming capability Flink is faster and better than Spark in streaming capability as Spark is near real time processing (handle stream in form of microbatching) but Flink is true real-time. Flink is the next generation big data framework, also known as '4G of big data', has been deployed in many Fortune 500 companies. Alibaba [31] is the typical case that works with buyers and suppliers through its web portal. Flink is being used by the company for online recommendations. When the acitivity 'Double Eleven day' comes, ringing up almost $2 billion in purchases in the buying frenzy's opening two minutes in 2018, Flink plays a key role in processing such massive momentary data.

5. Conclusion

Big data can be read about in almost any sphere. Through big data frameworks, organizations can now focus shifted towards incur huge savings, expedite and improve upon the daily procedures of your business, precisely target marketing, support intelligent, proactive and predictive businesses processes. A new series of execution frameworks have been added to the Hadoop ecosystem, including Spark, Flink, Storm and Samza, each with its own strengths and weaknesses.Hadoop is the first popular big data tool and fits batch-oriented applications while Samza, Storm only work for streaming. Spark and Flink are hybrid framework for both batch and streaming processing, but Spark is not true streaming process as on top of micro-batching. As of today, Flink is leading the big data field, excelling at more targeted tasks, as it contains nearly all the demanding functions including exactly once, high throughput, low latency, state management, fault tolerance and etc, though it will likely be adapted to a broader set of use cases in the future. When it comes to data analytics, one size doesn't fit all, and hence combining execution frameworks for most efficient performance is the silver bullet for every use case. Going forward, organizations need the option to incorporate new ones as they emerge in the big data landscape to embrace changing data volumes and future uses cases. The future is what we make it.

Word Count: 8228

References

[1] S. Ghemawat, H. Gobioff and S. Leung, "The Google file system", *ACM SIGOPS Operating Systems Review*, vol. 37, no. 5, p. 29, 2003. Available: 10.1145/1165389.945450.

[2] S. Dean and S. Ghemawat, "MapReduce", *Communications of the ACM*, vol. 51, no. 1, p. 107, 2008. Available: 10.1145/1327452.1327492.

[3] K. Shvachko, H. Kuang, S. Radia, and R.Chansler.The hadoop distributed file system. InMSST vol. 10, p. 1-10, 2010.

[4]2019. [Online]. Available: https://community.hortonworks.com/questions/51005/what-is-the-procedure-for-re-replication-od-lost-b.html. [Accessed: 08- May- 2019].

[5] G. Mackey, S. Sehrish, J. Bent, J. Lopez, S. Habib, and J. Wang, "Introducing map-reduce to high end computing", *2008 3rd Petascale Data Storage Workshop* p. 1-6, 2008.

[6] A. Talwalkar, "HadoopT-breaking the scalability limits of Hadoop", 2011

[7]Vavilapalli, K. Vinod, C. Arun, Murthy, C. Douglas, S. Agarwal, M. Konar, R. Evans, T. Graves, J. Lowe, H. Shah, S. Seth, and B. Saha, "Apache hadoop yarn: Yet another resource negotiator", In *Proceedings of the 4th annual Symposium on Cloud* Computing *of the ACM,* p.5, 2013.

[8] 2019. [Online]. Available: https://community.hortonworks.com/questions/51005/what-is-the-procedure-for-re-replication-od-lost-b.html. [Accessed: 08- May- 2019].

[9] T. White, *Hadoop*. Sebastopol, CA: O'Reilly, 2012, pp. 49-52.

[10] H. Karau, *Learning Spark*. Sebastopol: O'Reilly, 2015, pp. 23-29.

[11] "MLlib: Main Guide - Spark 2.4.3 Documentation", *Spark.apache.org*, 2019. [Online]. Available: https://spark.apache.org/docs/latest/ml-guide.html. [Accessed: 08- May- 2019].

[12] J. Scott, "A tale of two clusters: Mesos and YARN", *O'Reilly Media*, 2019. [Online]. Available: https://www.oreilly.com/ideas/a-tale-of-two-clusters-mesos-and-yarn. [Accessed: 08- May- 2019].

[13] D. Team, "Features of Apache Spark - Learn the benefits of using Spark - DataFlair", *DataFlair*, 2019. [Online]. Available: https://data-flair.training/blogs/apache-spark-features/. [Accessed: 08- May- 2019].

[14] J. Leibiusky, G. Eisbruch and D. Simonassi, Getting started with Storm. Sebastopol, CA: O'Reilly Media, 2012.

[15] M. Hussain Iqbal and T. Rahim Soomro, "Big Data Analysis: Apache Storm Perspective", International Journal of Computer Trends and Technology, vol. 19, no. 1, pp. 9-14, 2015. Available: 10.14445/22312803/ijctt-v19p103.

[16] S. Noghabi et al., "Samza", Proceedings of the VLDB Endowment, vol. 10, no. 12, pp. 1634-1645, 2017. Available: 10.14778/3137765.3137770.

[17] Noghabi, S. A., Paramasivam, K., Pan, Y., Ramesh, N., Bringhurst, J., Gupta, I., & Campbell, R. H. (2017). Samza: stateful scalable stream processing at LinkedIn. *Proceedings of the VLDB Endowment*, *10*(12), 1634-1645.

[18] Zhuang, Z., Feng, T., Pan, Y., Ramachandra, H., & Sridharan, B. (2016, June). Effective multi-stream joining in apache samza framework. In *2016 IEEE International Congress on Big Data (BigData Congress)* (pp. 267-274). IEEE.

[19] Carbone, P., Ewen, S., Fóra, G., Haridi, S., Richter, S. and Tzoumas, K. (2017). State management in Apache Flink®. Proceedings of the VLDB Endowment, 10(12), pp.1718-1729.

[20] Katsifodimos, P. C. A., Markl, S. E. V., & Tzoumas, S. H. K. (2015). Apache FlinkTM: Stream and Batch Processing in a Single Engine. Bull. IEEE Comput. Soc. Tech. Comm. Data Eng, 36(4).

[21] Perwej, Y., Kerim, B., Sirelkhtem, M., & Sheta, O. E. An Empirical Exploration of the Yarn in Big Data.

[22] Friedman, E., & Tzoumas, K. (2016). Introduction to Apache Flink: stream processing for real time and beyond. " O'Reilly Media, Inc.".

[23] J. Samosir, M. Indrawan-Santiago and P. Haghighi, "An Evaluation of Data Stream Processing Systems for Data Driven Applications", *Procedia Computer Science*, vol. 80, pp. 439-449, 2016. Available: 10.1016/j.procs.2016.05.322.

[24] Xie, W., Li, P., & Xu, H. (2018, July). Architecture and Implementation of Real-Time Analysis System Based on Cold Chain Data. In *Conference on Complex, Intelligent, and Software Intensive Systems* (pp. 497-505). Springer, Cham.

[25] Gurusamy, V., Kannan, S., & Nandhini, K. (2017). The Real Time Big Data Processing Framework: Advantages and Limitations. vol, 5, 305-312.

[26] Karim, S., Soomro, T. and Aqil Burney, S. (2018). Spatiotemporal Aspects of Big Data. Applied Computer Systems, 23(2), pp.90-100.

[27] Noghabi, S. A., Paramasivam, K., Pan, Y., Ramesh, N., Bringhurst, J., Gupta, I., & Campbell, R. H. (2017). Samza: stateful scalable stream processing at LinkedIn. Proceedings of the VLDB Endowment, 10(12), 1634-1645.

[28] Chandel, S. (2017). Comparative Study of Open Source Processing Frameworks for Analysis of Big Data. International Journal of Advanced Research in Computer Science, 8(5).

[29] "2018 Big Data Learning Getting Started Planning - Programmer Sought", *Programmersought.com*, 2019. [Online]. Available: http://www.programmersought.com/article/597354913/. [Accessed: 08- May- 2019].

[30] "Samza - Low Latency Web-Scale Fraud Prevention", *Samza.apache.org*, 2019. [Online]. Available: https://samza.apache.org/case-studies/ebay. [Accessed: 08- May- 2019].

[31] "A Flink Series from the Alibaba Tech Team", *Medium*, 2019. [Online]. Available: https://medium.com/@alitech_2017/a-flink-series-from-the-alibaba-tech-team-b8b5539fdc70. [Accessed: 08- May- 2019].