Proc. of Int. Conf. on Current Trends in Eng., Science and Technology, ICCTEST

Survey On Frameworks used in BigData Analytics

Yashaswini B M
CSE dept, Don Bosco Institutue of technology , Bangalore, India
yashu.giri2008@gmail.com@gmail.com

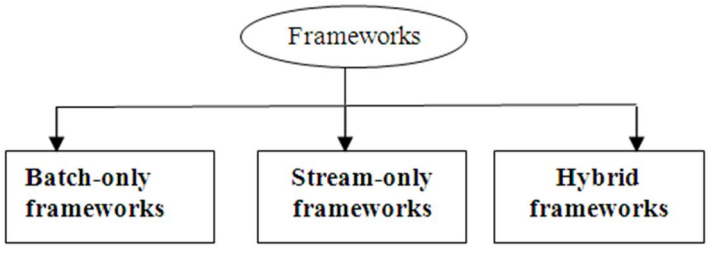
***Abstract*—Big data analytics is the procedure of inspecting huge data sets to expose hidden patterns, unidentified associations, market trends, purchaser preferences and other useful business information. i.e. data analytics helps organizations harness their data and use it to identify new opportunities. Examining big data allows analysts, researchers, and business operators to make better and faster decisions using data that was before difficult to get to or unfeasible. Using progressive analytics methods such as text analytics, machine learning, predictive analytics, data mining, statistics, and usual language processing, businesses can study before unused data sources independent or together with their existing enterprise data to gain new insights resulting in significantly better and faster decisions. Thus to analyze such a growing data certain frameworks can be used. In this paper a survey on all the frameworks to analyze data and the comparative analysis on them is carried out.**

***Keywords*— Bigdata;Framework;Hadoop; (key words).**

I. INTRODUCTION

BIG data is transforming business landscapes and the unprecedented excitement surrounding business analytics and big data has been generated primarily from the web and e-commerce communities [16]. Web-based systems (WBS), ranging from product recommender systems, e- commerce platforms, social networking, gambling, gaming, to CRM (Customer Relationship Management), and SCM (Supply Chain Management) applications, have tradition- ally relied on data analytics to operate. For WBS, data ana-lytics is about generating predictions and tortious visions to improve real-time customer experience, increase market and customer intelligence, predict customer behav-iors, optimize operational efficiency, personalize service provision, prevent security threats and frauds, minimize brand risk, and innovate processes and services

Categories of framework



.

Grenze ID: 02.ICCTEST.2017.1.129

Processing frameworks and processing engines are accountable for computing over information in a data system. Though there is no impressive definition setting apart "engines" from "frameworks", it is occasionally worthwhile to define the former as the actual component responsible for functioning on data and the latter as a set of components designed to do the same

Batch Processing Systems

Batch processing has a long history inside the big data world. Batch processing includes operating over a large , static dataset and returning the result at a later time when the computation is complete
The datasets in batch processing are typically...

- bounded: batch datasets denote a finite collection of data
- persistent: data is almost always supported by some type of permanent storage
- large: batch operations are often the only selection for processing extremely large sets of data

Batch processing is compatible for calculations where access to a complete set of archives is important. For example, when calculating totals and averages, datasets must be treated holistically instead of as a collection of individual records. These operations encompass that state be maintained for the time of the calculations.
Tasks that need huge capacities of data are often best handled by batch processes. Whether the datasets are treated directly from permanent storage or loaded into memory, batch systems are built with huge quantities in mind and have the resources to handle them. Because batch processing excels at handling huge volumes of determined data, it normally is used with historical data.
The trade-off for management of huge quantities of data is lengthier computation time. Because of this, batch processing is not suitable in situations where processing time is particularly important.

Type of batch processing system:Apache Hadoop

Apache Hadoop is a processing framework that completely gives batch processing. Hadoop was the original big data framework to advance important traction in the open-source community. Based on numerous documents and exhibitions by Google about how they were dealing with great volumes of data at the time, Hadoop re implemented the algorithms and component stack to make big scale batch processing more accessible.
Modern forms of Hadoop are collected of numerous components that work composed to process batch data:

- **HDFS**: HDFS is the distributed filesystem layer that coordinates storage and reproduction across the cluster nodes. HDFS confirms that data remains available in spite of inevitable host failures. It is used as the basis of data, to store intermediary processing outcomes, and to persevere the final calculated outcomes.
- **YARN**: YARN, which stands for yet additional Resource Negotiator, is the cluster managing component of the Hadoop stack. It is accountable for interacting and handling the underlying assets and scheduling jobs to be run. YARN makes it possible to run much more varied workloads on a Hadoop bunch than was possible in previous repetitions by acting as an interface to the cluster resources.
- **MapReduce**: MapReduce is Hadoop's inborn batch processing engine.

Batch Processing Model

The processing functionality of Hadoop originates from the MapReduce engine. MapReduce's processing technique follows the map, shuffle and decrease algorithm using key-value pairs. The basic process comprises:

- Interpret the dataset from the HDFS filesystem
- Separating the dataset into chunks and distributed among the available nodes
- Putting the calculation on to each node to the subset of data (the intermediary results are inscribed back to HDFS)
- Redistributing the intermediary results to group by key
- "Reducing" the value of each key by shortening and joining the results calculated by the separate nodes
- Write the calculated final results back to HDFS

Stream Processing Systems

Stream processing systems calculate above data as it enters the system. This needs a dissimilar processing model than the batch example. Instead of significant operations to apply to an entire dataset, stream processors define processes that will be applied to each separate data item as it passes through the system.

The datasets in stream giving out are considered "unbounded". This has a few significant implications:

- The *entire* dataset is only well-defined as the quantity of data that has arrived the system so far.
- The *working* dataset is perhaps more applicable, and is limited to a single item at a time.
- Processing is event-based and does not "stop" until explicitly stopped. Results are immediately available and will be continually updated as new data arrives.

Stream processing systems can handle a closely limitless amount of data, but they only process one (true stream processing) or very insufficient (micro-batch processing) items at a time, with minimal state being maintained in between records. Though most systems provide procedures of maintaining some state, steam processing is extremely optimized for more **useful processing** with few side effects.

Functional processes focus on separate steps that have limited state or side-effects. Performing the similar operation on the same part of data will give the same output independent of other factors. This kind of processing fits with streams as state between items is usually some combination of difficult, limited, and sometimes undesirable. So while some type of state management is typically possible, these frameworks are much simpler and more efficient in their absence.

This type of processing gives itself to some types of workloads. Processing with near real-time requirements is well served by the flowing model. Analytics, server or application error logging, and other time-based metrics are a natural fit because reacting to changes in these areas can be critical to business purposes. Stream processing is a decent fit for data where you must answer to changes or spikes .

Apache Storm

Apache Storm is a stream processing framework that focuses on tremendously low latency and is possibly the best option for workloads that require near real-time processing. It can handle very large quantities of data with and bring results with fewer latency than other solutions.

Stream Processing Model

Storm stream processing works by scoring DAGs (Directed Acyclic Graphs) in a framework it calls **topologies**. These topologies defines the numerous alterations or steps that have been taken from each inward part of data as it enters the system.

The topologies are made up of:

- **Streams**: Conventional data streams. This is unbounded data that is constantly arriving at the system.
- **Spouts**: Sources of data streams at the edge of the topology. These can be APIs, queues, etc. that gives data to be operated on.
- **Bolts**: Bolts denote a processing step that consumes streams, put on an operation to them, and provides the outcome as a stream. Bolts are associated to each of the spouts, and then link to each other to assemble all of the necessary processing. At the end of the topology, final bolt output may be used as an input for a linked system.

The purpose of Storm is to define small, distinct operations using the above components and then arrange them into a topology. By default, Storm provides at-least-one time processing guarantees, means that it can guarantee that every message is handled at least one time, but there may be duplicates in some failed scenarios. Storm does not provides that messages will be processed in order.

In order to achieve exactly-once, stateful processing, an abstraction called **Trident** is also available. To be clear, Storm without Trident is often referred to as **Core Storm**. Trident significantly alters the processing dynamics of Storm, increasing latency, adding state to the processing, and implementing a micro-batching model as an alternative of an item-by-item pure streaming system.

Storm users typically recommend using Core Storm whenever possible to avoid those penalties. With that in mind, Trident's promise to processes items exactly once is useful in cases where the system cannot intelligently handle duplicate messages. Trident is also the only option within Storm when you need to sustain state between items, like when counting how many users click a link within an hour. Trident gives Storm flexibility, although it does not play to the framework's natural strengths.

Trident topologies are composed of:

Stream batches:
These are micro-batches of stream data that are chunked in order to provide batch processing semantics.

Operations:
These are batch processs that can be performed on the data

Stream Processing Model
Samza relies on Kafka's semantics to define the way that streams are handled. Kafka uses the following concepts when handling the data:

- **Topics**: Each stream of data entering a Kafka system is called a topic. A topic is basically a stream of interrelated information that consumers can subscribe to.
- **Partitions**: In order to distribute a topic among nodes, Kafka divides the incoming messages into partitions. The partition divisions are based on a key such that each message with the same key is assured to be sent to the same partition. Partitions have guaranteed ordering.
- **Brokers**: The individual nodes that make up a Kafka cluster are called brokers.
- **Producer**: Any aspect writing to a Kafka topic is called a producer. The producer provides the key that is used to partition a topic.
- **Consumers**: Consumers are any component that reads from a Kafka topic. Consumers are accountable for maintaining information about their own offset, so that they are aware of which records have been processed if a failure occurs.

Since Kafka is represents an immutable log, Samza deals with immutable streams. This means that any transformations create new streams that are consumed by other components without affecting the initial stream.
Samza is able to store state, using a fault-tolerant checkpointing system implemented as a local key-value store.

Hybrid Processing Systems: Batch and Stream Processors

Some processing frameworks can manage both batch and stream workloads. These frameworks simplify diverse processing requirements by allowing the same or related components and APIs to be used for both types of data.
As you will see, the way this is achieved varies significantly between Spark and Flink, the two frameworks we will discuss. This is a mainly a purpose of how the two processing paradigms are brought jointly and what assumptions are made about the relationship between fixed and unfixed datasets.
While projects focused on one processing type may be a close fit for precise use-cases, the hybrid frameworks attempt to offer a general solution for data processing. They not only provide methods for processing over data, they have their own libraries, and tooling for doing things like, machine learning, and interactive querying.

Apache Spark

Apache Spark is a next generation batch processing framework with stream processing capabilities. Built using many of the similar values of Hadoop's MapReduce engine, Spark focuses primarily on speeding up batch processing assignments by presents full in-memory computation and processing optimization.
Spark can be deployed as a separate cluster (if paired with a capable storage layer) or can hook into Hadoop as an alternative to the MapReduce engine.

Batch Processing Model
Unlike MapReduce, Spark processes all data in-memory, only interacting with the storage layer to initially load the data into memory and at the end to keep on the final results. All intermediary results are managed in memory.
While in-memory processing contributes considerably to speed, Spark is also speeder on disk-related assignments because of holistic optimization that can be achieved by analyzing the whole set of assignments ahead of time. It reaches this by creating Directed Acyclic Graphs, or **DAGs** which signify all of the processes that must be performed, the data to be operated on, as well as the relationships between them, giving the processor a greater capability to logically coordinate work.
To implement in-memory batch computation, Spark uses a model called Resilient Dispersed Datasets, or **RDDs**, to work with data. These are absolute structures that be within memory that represent collections of

data. Processes on RDDs produce new RDDs. Each RDD can trace its lineage back through its parent RDDs and ultimately to the data on disk. Essentially, RDDs are a way for Spark to keep fault tolerance without needing to write back to disk after each operation.

Stream Processing Model
Stream processing capabilities are given by Spark Streaming. Spark itself is designed with batch-oriented assignments in mind. To deal with the disparity between the engine design and the characteristics of streaming assignments, Spark implements a concept called *micro-batches**. This strategy is designed to treat streams of data as a series of very small batches that can be managed using the natural semantics of the batch engine.
Spark Streaming works by buffering the stream in sub-second increments. These are sent as small fixed datasets for batch processing. In practice, this jobs honestly well, but it does lead to a different performance profile than true stream processing frameworks.

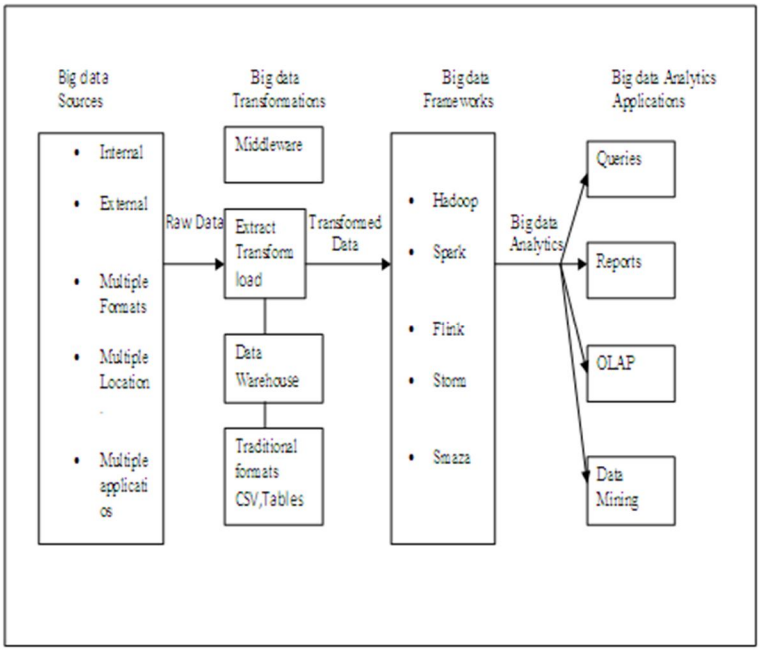
Apache Flink
Apache Flink is a stream processing framework that can also manage batch assignments. It considers batches to simply be data streams with finite borders, and thus treats batch processing as a subset of stream processing. This stream-first approach to all processing has a numeral of interesting side effects.
This stream-first approach has been called the **Kappa architecture**, in difference to the more widely known Lambda architecture (where batching is used as the primary processing method with streams used to supplement and provide early but unrefined results). Kappa architecture, where streams are used for all, shortens the model and has only recently become possible as stream processing engines have grown more sophisticated.

Stream Processing Model
Flink's stream processing model handles inward data on an item-by-item basis as a true stream. Flink gives its DataStream API to work with unbounded streams of data. The basic parts that Flink works with are:

- **Streams** are immutable, limitless datasets that flow through the system
- **Operators** are functions that function on data streams to produce extra streams
- **Sources** are the entrance point for streams ingoing the system
- **Sinks** are the place where streams movement out of the Flink system. They might signify a database or a connector to another system
- Stream processing tasks take photos at set points during their computation to use for recovery in case of difficulties. For storing state, Flink can work with a number of state back ends dependent with varying levels of complexity and persistence.
- Moreover, Flink's stream processing is capable to know the concept of "event time", meaning the time that the event really happened, and can handle sessions as well. This means that it can guarantee ordering and grouping in some interesting ways.

Batch Processing Model
Flink's batch processing model in many ways is just an postponement of the stream processing model. In its place of reading from a continuous stream, it reads a bounded dataset off of persistent storage as a stream. Flink uses the exact same runtime for both of these processing models
Flink offers more or less optimizations for batch assignments. For instance, since batch operations are supported by persistent storage, Flink eliminates snapshotting from batch loads. Data is still recoverable, but normal processing finishes faster.
Alternative optimization involves breaking up batch tasks so that stages and components are only involved when required. This helps Flink play well with other users of the cluster. Preemptive analysis of the assignments gives Flink the capability to also enhance by seeing the whole set of operations, the size of the data set, and the requirements of steps coming down the line.

II. TYPES OF FRAME WORKS



Hadoop Framework

Working with the Hadoop framework requires new big data solutions.
Apache Hadoop is a critical technology for any enterprise require to take benefit of large data. The Hadoop framework brings the processing power big data needs by issuing the processing of massive data sets across multiple computer clusters – scaling from one server to thousands as needed. It arrange for high availability by managing failure at the application layer, and continuing processes even when a single server or cluster fails. And the Hadoop framework provides the efficiency needed to process big data by not asking applications to move large capacities of data across a network.

HDFS and MapReduce
There are two main components at the core of Apache Hadoop 1.x: the Hadoop Distributed File System (HDFS) and the MapReduce similar processing framework. These are both open source projects, encouraged by technologies created inside Google.

Apache Storm

Storm makes it easy to consistently process limitless streams of data, doing for real-time processing what Hadoop did for batch processing. Storm is simple, can be used with any programming language.
Storm has many use cases: real-time analytics, online machine learning, non-stop computation, distributed RPC, ETL, and more. Storm is fast: a benchmark clocked it at over **a million tuples processed per second per node**. It is scalable, fault-tolerant, guarantees your data will be processed, and is easy to set up and operate.

Apache Spark

Apache Spark is a fast, in-memory data processing engine with graceful and communicative development APIs to permit data workers to efficiently execute streaming, machine learning or SQL assignments that require fast iterative access to datasets. With Spark running on Apache Hadoop YARN, designers everywhere can now create applications to exploit Spark's power, derive insights, and enrich their data science assignments within a single, communal dataset in Hadoop.

Apache Flink
Apache Flink is a distributed data processing stage for use in big data applications, mainly involving analysis of data stored in Hadoop clusters. Backup a mixture of in-memory and disk-based processing, Flink manages

both batch and stream processing jobs, with data streaming the default implementation and batch jobs running as special-case forms of streaming applications.
Flink was designed as an other to MapReduce, the batch-only processing engine that was paired with the Hadoop Distributed File System (HDFS) in Hadoop's initial incarnation.

Comparative analysis of the frameworks

Frame works	Features
Apache Flink	Flink is a framework for unified stream and batch processing Flink has a record-based or any custom user-defined Window criteria.
Apache Spark	Spark is based on micro-batch modal. Spark has a time-based Window criteria
Apache Storm	Supports true streaming processing model through core strom layer Storm provides configuring initial parallelism at various levels per topology
Hadoop	Hadoop is highly scalable in the way new hardware can be easily added to the nodes. Data is highly available and accessible despite hardware failure due to multiple copies of data.
Samza	Samza provides a very simple callback-based “process message” API Samza uses Kafka to guarantee that messages are processed in the order they were written to a partition, and that no messages are ever lost.

III. CONCLUSION

Hadoop and Spark frameworks are the utmost aware and utmost implemented of the projects in the space. They are also mainly batch processing frameworks. The final 3 frameworks are all real-time or real-time-first processing frameworks; as such, this post does not significance to be an apples-to-apples evaluation of frameworks. Instead, these numerous frameworks have been presented to get to know them a bit better, and understand where they may fit in.

REFERENCES

[1] Katal, A., Wazid, M., Goudar, R.H., "Big data: Issues, challenges, tools and Good practices", Sixth International Conference on Contemporary Computing (IC3) 2013.
[2] Stephen K, Frank A, J. Alberto E, William M, "Big Data: Issues and Challenges Moving Forward", IEEE, 46th Hawaii International Conference on System Sciences, 2013.
[3] Sachchidanand S, Nirmala S, "Big Data Analytics", IEEE, International Conference on Communication, Information & Computing Technology (ICCICT), Oct. 19-20, 2012.
[4] Katina Michael, Keith W. Miller, "Big Data: New Opportunities and New Challenges", IEEE Technology and Society Magazine, vol 13.

[5] Jeffrey Dean and Sanjay Ghemawat, "MapReduce: Simplified Data Processing on Large Clusters", Sixth Symposium on Operating System Design and Implementation, San Francisco, CA, December, 2004.
[6] "Apache-Hadoop"-http://hadoop.apache.org/ #What+Is+Apache+Hadoop%3F