IJCSI International Journal of Computer Science Issues, Volume 14, Issue 5, September 2017
ISSN (Print): 1694-0814 | ISSN (Online): 1694-0784
www.IJCSI.org https://doi.org/10.20943/01201705.612

Real-time Data Stream Processing Challenges and Perspectives

OUNACER Soumaya [1], TALHAOUI Mohamed Amine [2], ARDCHIR Soufiane [3], DAIF Abderrahmane[4] and AZOUAZI Mohamed[5]

[1] Hassan II University, Faculty Of Sciences Ben m'Sik, Laboratoire Mathématiques Informatique et Traitement de l'Information MITI, Casablanca, Morocco

[2] Hassan II University, Faculty Of Sciences Ben m'Sik, Laboratoire Mathématiques Informatique et Traitement de l'Information MITI, Casablanca, Morocco

[3] Hassan II University, Faculty Of Sciences Ben m'Sik, Laboratoire Mathématiques Informatique et Traitement de l'Information MITI, Casablanca, Morocco

[4] Hassan II University, Faculty Of Sciences Ben m'Sik, Laboratoire Mathématiques Informatique et Traitement de l'Information MITI, Casablanca, Morocco

[5] Hassan II University, Faculty Of Sciences Ben m'Sik, Laboratoire Mathématiques Informatique et Traitement de l'Information MITI, Casablanca, Morocco

Abstract

Nowadays, with the recent evolution of sensor technologies, wireless communications, powerful mobile devices and other real-time sources, the way to process the high-speed and real-time data stream brings new challenges.

These challenges of big data systems are to detect, anticipate and predict information with the finest granularity possible. The problem is that the system relies on batch processing, which can give great insight into what has happened in the past; however, they do not have the capacity to deal with what is happening at the moment, and of course it is crucial to process events as they happen to a real-time preview. Many applications like fraud detection, data analytics, and production inspection need a fast response and processing time since big data system is based on the MapReduce framework that can only process a finite set of data, is not suitable for the processing of data stream and is inappropriate to satisfy the constraints in real time. Hence the need for a real-time data stream processing system, since it is fast and processes the data in a minimal time with low latency.

This paper gives a clear comparison among the different systems that exist for real time data stream processing as well as a model that was based on the comparison that was conducted before.

Keywords: *Real-time processing, MapReduce, Spark, Storm, Lambda architecture, kappa architecture.*

1. Introduction

Nowadays, the world we live in generates a large volume of information and data from different sources, namely search engines, social networks, computer logs, e-mail clients, sensor networks...etc All of these data are called masses of data or Big Data. For instance, a minute generates 347,222 new tweets on Twitter, about 701,389 Facebook logins, more than 2.78 million videos views on YouTube, and 20.8 million messages on WhatsApp… etc[1]. All these data are generated continuously and in the form of streams.

The recent evolution of sensor technologies, wireless communications, as well as powerful mobile devices are all under the umbrella of applications of internet of things, and the way to process the high-speed and real-time data stream brings new challenges. The new challenges of big data systems today are to detect, anticipate and predict information with the finest granularity possible. The problem is that the system relies on batch processing [2] which can give great insight into what has happened in the past; however, they do not have the capacity to deal with what is happening at the moment, and of course it is crucial to treat events as they happen to a real-time preview. Big data is based on the MapReduce framework that can only process a finite set of data, is not suitable for the processing of data stream and is inappropriate to satisfy the constraints in real time[3]. Hence the need for a real time data stream processing system, since it is fast and processes the data in a minimal time with low latency.

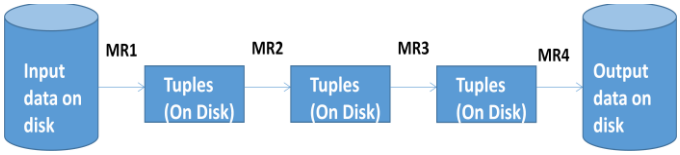

Fig. 1 Mapreduce Jobs

Mapreduce is fundamentally suitable for parallelize processing on a large amount of data, but it's not the best tool for processing the latest version of data. This framework is based on disk approach and each iteration output is written to disk making it slow. Figure 1 represents MapReduce jobs; MapReduce reads data from the disk and writes them again in the disk four times which means that the complete flow becomes very slow which degrades the performance.

The rest of this paper is organized as follows: in section II, we define the basics of big data, ecosystem and stream processing. In section III, we present a survey of data processing tools. In section IV, we focus on a comparative study of the different systems of processing of data stream. In section V, we present an overview of two real time processing architectures. And last but not least, in section VI we suggest a model that was based on the previous comparisons.

2. Big data: Theoretical Foundation

This section is devoted to some of the main concepts used in big data including an introduction of big data, its architecture, technologies used and concepts on big data stream.
Big data is a new concept which has been introduced due to the large volume and complex data that become difficult to process using traditional data base methods and tools. According to Gartner [4] "Big data is high volume, high-velocity and high-variety information assets that demand cost-effective, innovative forms of information processing for enhanced insight and decision making." In 2010, [5] Chen et al. defined big data as "datasets which could not be captured, managed, and processed by general computers within an acceptable scope." [6]NIST says that "Big data shall mean the data of which the data volume, acquisition speed, or data representation limits the capacity of using traditional relational methods to conduct effective analysis or the data which may be effectively processed with important horizontal zoom technologies". The characteristics of big data are summarized in the five Vs: Volume, Velocity, Variety, Veracity and Value. Volume represents the size or the quantity of the data from terabyte to yotabyte. It is a massive evolution we are talking about, since 2005 the data were limited to 0.1 ZB, and they may reach 40 ZB and more in 2020[7]. Velocity means that the data must be processed and analyzed quickly in terms of the speed of their capture. Variety indicates that the data are not of the same type, which allows us to harness different types of data structured, semi-structured and non-structured.
Veracity targets the confidence in the data on which decisions are based. Last but not least, Value which means that systems must not only be designed to process massive data efficiently but also be able to filter the most important data from all collected data.
According to previously stated definitions, we can say that big data is an abstract concept, which makes it possible to extract the following problems: how to store, analyze, process and extract the right information from a varied datasets quickly generated and in the form of a data stream.

Stream processing is a technology that enables the data to be collected, integrated, analyzed and visualized in real time while the data is being produced[8]. Stream processing solutions are designed to handle big data in real time with a highly scalable, highly available, and highly fault tolerant architecture. This empowers to analyze the data in motion[9].
The goal of real time processing is to provide solutions that can process continuous infinite stream of data integrated from both live and historical sources in very fast and interactive way.

3. Data stream processing tools

Ancient methods used to process data, including Hadoop precisely MapReduce jobs, are not adequate for real time processing. Real time data stream processing keeps you up to date with what is happening at the moment whatever is the speed or the volume of data needless of the storage system. In order to understand well the system at hand, we are going to present a brief overview of the other platforms namely Hadoop, Spark, as well as Storm.

3.1 Apache Hadoop

The Apache Hadoop [10] is a software that is open source used to process big data across clusters of machines and operate these sets of data in batches. The heart of Hadoop is divided in two main parts namely MapReduce for processing data, and HDFS for storing data. It is known for its reliability, scalability and its processing model.
MapReduce was first introduced by Jeffrey Dean and Sanjay Ghemawat at Google in 2004[11], it is a programming model and an associated implementation for processing and generating large data sets on large clusters of commodity of machines. It is highly scalable, it can process petabytes of data stored in HDFS on one cluster, and it is highly fault tolerant which lets you run programs on a cluster of commodity server. This framework is based on two servers, a master Job Tracker that is unique on the cluster, it receives MapReduce tasks to run and organize their execution on the cluster. It is also responsible for scheduling the jobs' component tasks on the slaves as well as monitoring them and re-executing the failed tasks. The other server is the Task Tracker, there are several per cluster, it performs the job MapReduce itself. Each one of the Task Trackers is a unit of calculation of the cluster.
Users specify a map function that processes a key/value pairs to generate a set of intermediate key/value pairs, and a reduce function that merges all intermediate values associated with the same intermediate key. As figure 2 shows, first the MapReduce library in the user program splits the input files into M pieces of typically 16-64MB per piece, the master picks idle workers and assigns each one a map task or a reduce task. A worker who is assigned a map task reads the contents of the corresponding input split. The intermediate key/value pairs produced by the map function are buffered in memory. Periodically, the buffered pairs are written to local disk. When a reduce worker has read all intermediate data for its partition,

it sorts it by the intermediate keys so that all occurrences of the same key are grouped together. The sorting is needed because typically many different keys map to the same reduce task. If the amount of the intermediate data is too large to fit in memory, an external sort is used. The reduce worker iterates over the sorted intermediate data and for each unique intermediate key encountered, it passes the key and the corresponding set of intermediate values to the user's reduce function[11].

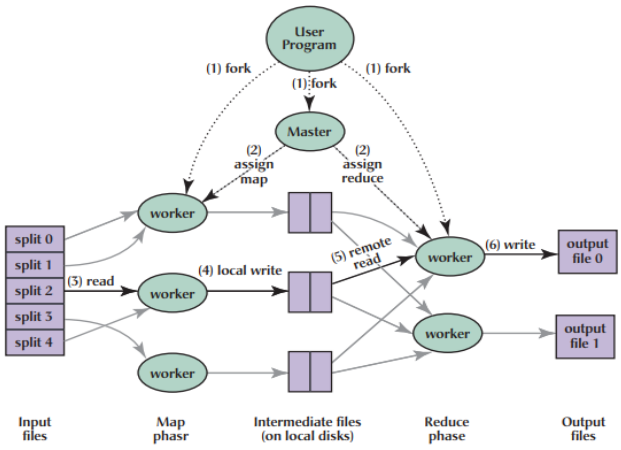

Fig. 2 Map Reduce Execution Overview

MapReduce has some limitations[12]:

- Developed for batch processing
- Based on disk approach
- Extremely rigid data flow
- Disk seek for every access
- Inefficient for iterative algorithms and interactive data mining

3.2 Apache Spark

Apache Spark[13]is an open source framework of big data processing built at the base of Hadoop MapReduce to perform sophisticated analysis and designed for speed and ease of use. This was originally developed by UC Berkeley University in 2009 and passed open source as an Apache project in 2010. Spark has lightning fast performance and speed up processing times because it runs in-memory on clusters. It is also designed to operate on batches like apache Hadoop, but the size of batch window is very small. The core of apache spark is RDD (Resilient Distributed Dataset), it is fault tolerant collection of elements distributed across many servers on which we can perform parallel operations. [14] The elements of an RDD need not exist in physical storage; instead, a handle to an RDD contains enough information to compute the RDD starting from data in reliable storage. This means that RDDs can always be reconstructed if nodes fail. Alongside Spark's main APIs, the ecosystem contains additional libraries that enable work in the field of big data analysis and machine learning. These libraries include spark streaming for processing a continuous data stream, Spark SQL for working with structured data, MLlib is a machine learning library, and GraphX for graph computation, as shown in figure 3.

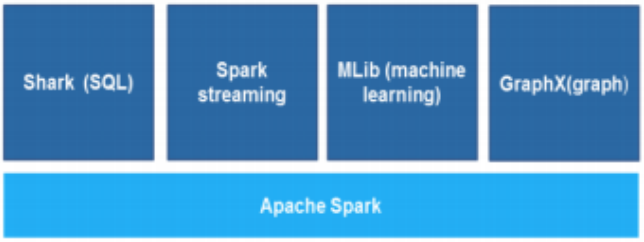

Fig. 3 Spark framework

Spark streaming [13]is an extension of the core Spark API that enables scalable, high-throughput, and fault-tolerant stream processing of live data streams. Data can be ingested from many sources like Kafka, Flume, Kinesis, or TCP sockets, and can be processed using complex algorithms expressed with high-level functions like map, reduce, join and window. [15]In Figure 4, we can see that spark streaming is based on a mode of processing in micro-batch It receives live input data streams and divides the data into batches, which are then processed by the Spark engine in a very fixed amount of time to generate the final stream of results in batches. All input streams are dealt with in the same way. The same recurrent timer allocates batches for all streams on every batch duration.
Spark has several advantages over Hadoop MapReduce. First, Spark offers a comprehensive and unified framework to meet the needs of big data processing. Then, Spark allows applications on Hadoop clusters to be run up to 100 times faster in memory and 10 times faster on disk. In addition, it is possible to use it interactively to query the data from a shell command window. Despite all the advantages of spark compared to Hadoop MapReduce, it remains confronted with several limitations; among them is the real time stream processing that is not ensured. This is due to the fact that spark implements the concept of micro-batches in its operation and does not process the data as they arrive because they are accumulated for a period of time before being processed. The major difference between Spark and Storm shall be discussed in upcoming sections.

Fig. 4 Spark streaming

3.3 Apache Storm

Storm, which [16] is a technology that was realized by Nathan Marz for real-time analysis in December 2010, is a free and open source distributed real time computation, and makes it easy to reliably process unbounded streams of data. Storm does for real time processing what Hadoop does for batch

processing; it's simple and can be used with any programming language.

A storm cluster has three sets of nodes, the first one is a daemon process called "Nimbus" similar to the Hadoop Job Tracker. It is running on main node in order to upload computations for execution, distribute codes across the cluster, arrange tasks and detect errors. The second node is called "Supervisor", it is responsible for starting and stopping the working process according to signals from Nimbus. And finally, the "Zookeeper" node which is a distributed coordination service that coordinates the storm cluster as shown in figure 5. A Storm cluster is superficially similar to a Hadoop cluster. Whereas on Hadoop you run "MapReduce jobs" on Storm you run "topologies". "Jobs" and "topologies" themselves are very different -- one key difference is that a MapReduce job eventually finishes, whereas a topology processes messages forever (or until you kill it).

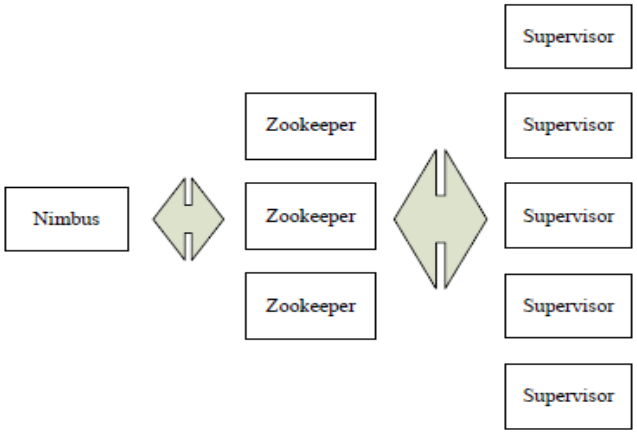

Fig. 5 Structure of Storm Cluster

A topology consists of spouts and bolts and the links between them show how streams are passing around. This topology is represented like a data processing Directed Acyclic Graph (DAG) which represents the whole stream processing procedure. A topology representation is shown below in figure 6.

A spout is a source of streams that reads tuples from external input source and emits them to the bolts as a stream. A bolt is a data processing unit of a storm topology which consumes any number of input streams, conducts some specific processing, and emits new streams out to other bolts. The core abstraction in Storm is the "stream". A stream is an unbounded sequence of tuples. A tuple is a named list of values, and a field in a tuple can be an object of any type. Storm provides the primitives for transforming a stream into a new stream in a distributed and reliable way.

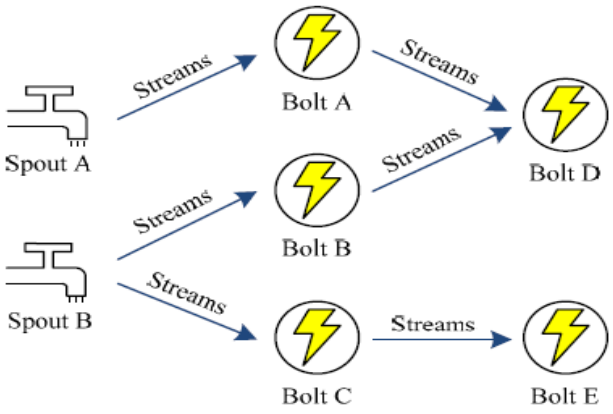

Fig. 6 Example of Topology

4. A Comparison of real time processing systems

In this section we compare the different tools used for the real time stream processing and according to this comparison we will choose the most suitable tool.

Table 1: Comparison of Big data processing tools

	Hadoop	**Spark**	**Storm**
Source Model	Open source	Open source	Open source
Architecture	Master/slaves	Master/slaves	Peer
Processing Model	Batch	Micro-batch	Real-time(one-at-a time)
Big data processing	Batch	Batch and Stream	Stream
achievable latency	High	A few seconds (< 1s)	Less than a second (< 100ms)
API Programmation	Java-Python and Scala	Java-Python and Scala	Any PL
Guaranteed Data Processing	exactly-once	exactly-once	At least once processing
Storage data	yes	yes	No
In memory	No	Yes	Yes
Fault tolerance	Yes	Yes	Yes
Ordering guarantees	Yes	Yes	No
Coordination tool	Zookeeper	Zookeeper	Zookeeper

The comparison above shows that storm is the best tool for real time stream processing, Hadoop does batch processing, and spark is capable of doing micro-batching. Storm uses the spouts and bolts to do one-at-a-time processing to avoid the inherent latency overhead imposed by batching and micro-batching.

5. Real time processing architectures

In this paper, we present a short overview of some of the Real time processing architectures namely Lambda and Kappa.

5.1 Lambda Architecture

The lambda architecture has been proposed by Nathan Marz. This architecture mixes the benefit of processing models, batch processing and real time processing to provide better results in low latency.

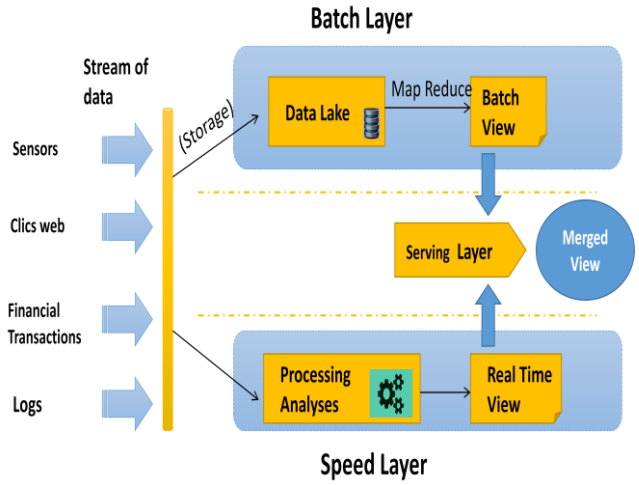

Fig. 7 Lambda architecture[17]

Figure 7 shows the basic architecture of lambda [18]. It is divided into three layers:

- **Batch layer** - manages historical data and re-computing results.
- **Speed layer** - receives the arriving data and performs incremental updates to the batch layer results.
- **Service layer** - enables various queries of the results sent from the batch and speed layers.

All new data are sent to both the batch and the speed layer. The batch layer is responsible for storing the master data set and contiguously computes views of these data with the use of the MapReduce algorithm. The results of the batch layer are called "batch views".
The serving layer indexes the pre-computed views produced by the batch layer. It is a scalable database that swaps in new batch views as they are made available. Due to the latency of the batch layer, the results available from the serving layer are always out of date by a few hours. The serving layer can be implemented using NoSQL technologies such as HBase, Apache Druid… etc.
The speed layer compensates for the high latency of updates to the serving layer. The role of this layer is to compute in real time the data that have not been taking into account in the last batch of the batch layer. It produces the real-time views that are always up to date and stores them in a fast store. The speed layer can be realized with data streaming technologies such as Apache Storm or Spark Streaming.

Yet, the lambda architecture has some limitations; the first thing is the business logic which is implemented twice in the real time and batch layers. The developers need to write the same code on both layers. The second remark consists of the need of more frameworks to master. And finally, there are simpler solutions when the need is less complex.

5.2 Kappa Architecture

Kappa architecture [19] as described by Jay kreps at Linkedin in 2014, is a software architecture pattern. Kappa is a simplification of lambda architecture which means it's like a Lambda Architecture system with the batch processing system removed. [19] The canonical data store in a Kappa Architecture system is an append-only immutable log. From the log, data is streamed through a computational system and fed into auxiliary stores for serving.

In fact, and even more than the lambda architecture, the Kappa architecture does not allow for the permanent storage of data. It is more dedicated to their processing. Although more restricted, the Kappa architecture leaves some freedom in the choice of components implemented.
In contrast to lambda architecture, which utilized two different code paths for the batch and the speed layer, Kappa uses only a single code path for the two layers which reduces system complexity[20]. The benefit of Kappa architecture is permitting users to develop, test, debug and operate their systems on top of a single processing framework. The figure below represents the Kappa architecture:

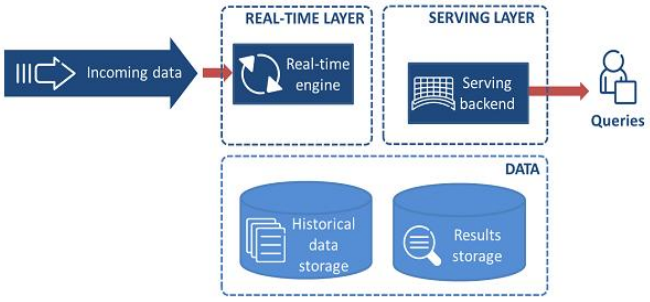

Fig. 8 Kappa architecture [21]

The chart below represents a short comparison of the two architectures as has been discussed before, namely Lambda and Kappa, following specific criteria.

Table 2: A comparison of real time processing architectures

	Lambda architecture	Kappa architecture
Architecture	immutable	immutable
Fault tolerance	yes	yes
Scalability	yes	yes
permanent storage	yes	no
Layers	Batch, real-time and service layer	Real-time and service layer
Processing data	Batch and streaming	streaming
Processing guarantees	Yes in batch but approximate in streaming	Exactly once
Re-processing pradigm	In every batch cycle	Just when code change
Real time	Isn't accurate	accurate

6. Proposed Architecture

According to the architectures and platforms presented in the previous paragraphs, we have presented the different benefits and disadvantages of each of these architectures, and according to its information, we designed a new architecture that is open source, and takes into account several criteria, among which the real-time processing of large data from high speed. It also allows an unlimited number of users to create many new and innovative features and make several improvements.

This architecture must ingest-filter-analyze and process incoming data streams with low latency, so the system must respond fairly quickly and it depends on the processing architecture used (spark, storm, etc.) or the size of the data and the complexity of the calculations performed. On the other hand, one must consider how to choose the most efficient tool; it should be easy to use not to pose to users, be it analysts or developers, infrastructure problems.

Perfectly, we want an architecture that allows making a transition to scale fairly easy and visually changing resource allocation. Furthermore, the newly configured resources have to join the cluster seamlessly and can handle changes in load or traffic without interrupting the streaming data processing globally.

And finally, a real-time architecture must provide a live streaming data visualization. It must allow the dynamic creation of dashboards, custom graphics, and UI extensions.

Figure 8 represents the new architecture, subdivided as a result:

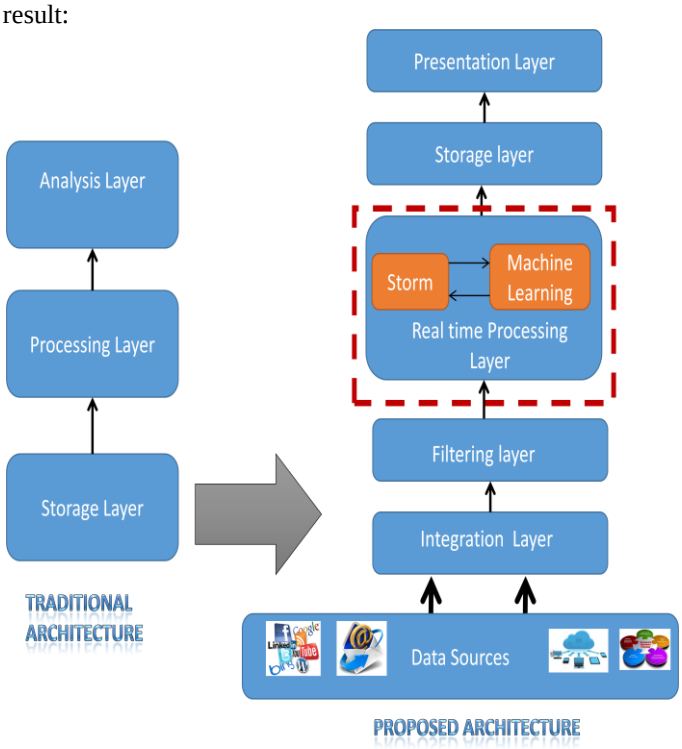

Fig. 9 Proposed architecture

Figure 8 represents both the traditional architecture of big data as well as the proposed architecture. The traditional architecture contains three layers namely storage, processing, and analysis, whereas our proposed architecture is represented as follows. The data come from different devices and equipments such as sensors, networks, cyber infrastructure, web, email, social media and many more. These Data, which come as a stream from different sources with a high speed, are acquired by the ***Integration Layer*** using a set of tools and functionalities (e.g. Apache Kafka). After being ingested, the data are going to be filtered through (ELT) extract-transform-load operations (e.g. PIG). In other words, the data are going to be cleaned, and their qualities are going to be analyzed …etc. This ***Filtering Layer*** serves the data to be prepared for the ***Real Time Processing Layer***, this latter aims to process the data in real time and with very low latency. As shown in figure 9, two technologies are to be used in this layer namely Storm, which is a tool for real time processing, and Machine Learning. The use of Machine Learning in this layer allows the archiving of data. Its goal is to visualize previous trends using a request/response method on similar inputs. ML learns continuously from new coming data which facilitates processing. Storm, on the other hand, is also used in this layer in order to process data in real time. It uses the notion of topology which is a network of Spout and Bolt. As has been noted before, the streams come from Spout that broadcasts data coming from external sources in Storm topology.

Concerning Bolts, we can implement some functionalities such as functions, filters, joins, aggregations…etc. So, Map function can be implemented in Bolt so as to mark the words of the stream. The resulting stream coming from Bolt 'Map' flows into the following Bolt that implements the 'Reduce' function in order to aggregate words into numbers.

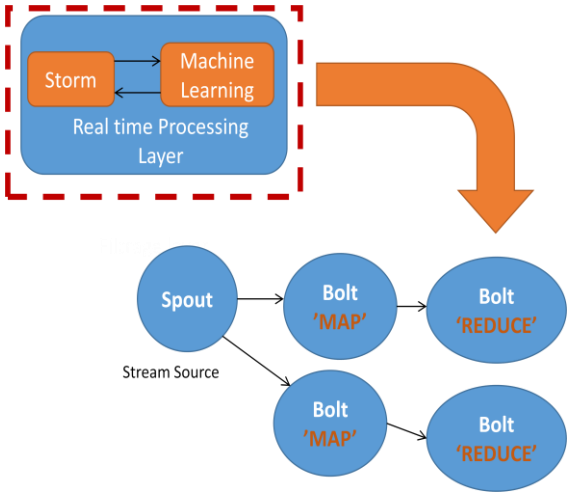

Fig. 10 Real Time Processing Layer

After the end of the processing, data are going to be stored either in a NoSQL base, or directly in the Distributed File system like HDFS, and finally the ***visualization layer*** will present to the user the result of the final data in streaming mode.

7. Conclusion

In this paper, we tried to present a state of the art concerning different concepts which led to conducting a thorough comparison of data stream processing tools. The main objective behind this comparison is to show that big data architecture is based on Batch processing which cannot process data in real time. Through this thorough comparison, storm was chosen as a tool for data processing because it is an open source that allows a real time processing with a very low latency.

Another comparison for real time processing architectures was also conducted so as to suggest a new architecture in which Storm and Machine Learning were used in order to facilitate the processing in real time.

Our next target is to implement and test this proposed architecture in the upcoming research.

References

[1] K. LEBOEUF, "2016 Update_ What Happens in One Internet Minute_ - Excelacom, Inc." [Online]. Available: http://www.excelacom.com/resources/blog/2016-update-what-happens-in-one-internet-minute.

[2] "MapReduce." [Online]. Available: https://fr.hortonworks.com/apache/mapreduce/.

[3] D. S. Terzi, U. Demirezen, and S. Sagiroglu, "Evaluations of big data processing," vol. 3, no. 1, 2016.

[4] Gartner Inc., "What Is Big Data? - Gartner IT Glossary - Big Data," *Gartner IT Glossary*. p. 1, 2013.

[5] M. Chen, S. Mao, and Y. Liu, "Big data: A survey," *Mob. Networks Appl.*, vol. 19, no. 2, pp. 171–209, 2014.

[6] G. Li, *Big data related technologies, challenges and future prospects*, vol. 15, no. 3. 2015.

[7] G. Herber, "Innovation Session Unlocking the Massive Potential of Sensor Data and the Internet of Things," 2014.

[8] M. M. Maske and P. Prasad, "A real time processing and streaming of wireless network data using Storm," *2015 Int. Conf. Comput. Power, Energy, Inf. Commun.*, pp. 0244–0249, 2015.

[9] K. Wähner, "Real-Time Stream Processing as Game Changer in a Big Data World with Hadoop and Data Warehouse," *InfoQ*. pp. 1–9, 2014.

[10] W. Is *et al.*, "Welcome to Apache™ Hadoop™!," *Innovation*, no. November 2008. pp. 2009–2012, 2012.

[11] J. Dean and S. Ghemawat, "MapReduce: Simplified Data Processing on Large Clusters," *Proc. 6th Symp. Oper. Syst. Des. Implement.*, pp. 137–149, 2004.

[12] G. C. Deka, "Handbook of Research on Cloud Infrastructures for Big Data Analytics," *Handbook of Research on Cloud Infrastructures for Big Data Analytics*. pp. 370–391, 2014.

[13] Apache Spark, "Apache Spark™ - Lightning-Fast Cluster Computing," *Spark.Apache.Org*. 2015.

[14] M. Zaharia, M. Chowdhury, M. J. Franklin, S. Shenker, and I. Stoica, "Spark : Cluster Computing with Working Sets," *HotCloud'10 Proc. 2nd USENIX Conf. Hot Top. cloud Comput.*, p. 10, 2010.

[15] A. Ghaffar, S. & Tariq, R. Soomro, A. G. Shoro, and & Tariq, "Big Data Analysis: Ap Spark Perspective," *Glob. J. Comput. Sci. Technol. Glob. Journals Inc. Glob. J. Comput. Sci. Technol.*, vol. 15, no. 1, pp. 7–14, 2015.

[16] N. Marz, "Tutorial." [Online]. Available: http://storm.apache.org/releases/1.1.0/Tutorial.html.

[17] "Lambda Architecture," 2014. [Online]. Available: http://lambda-architecture.net/.

[18] N. Marz, *Big Data - Principles and best practices of scalable realtime data systems*. 2012.

[19] "Kappa Architecture - Where Every Thing Is A Stream." .

[20] J. Kreps, "Questioning the Lambda Architecture," *O'Reilly*. pp. 1–10, 2014.

[21] J. Forgeat, "Data processing architectures – Lambda and Kappa," *Ericsson Research Blog*. 2015.