A New Architecture for Real Time Data Stream Processing

Soumaya Ounacer, Mohamed Amine TALHAOUI, Soufiane Ardchir, Abderrahmane Daif and Mohamed Azouazi
Laboratoire Mathématiques Informatique et Traitement de l'Information MITI
Hassan II University, Faculty Of Sciences Ben m'Sik Casablanca, Morocco

***Abstract*—Processing a data stream in real time is a crucial issue for several applications, however processing a large amount of data from different sources, such as sensor networks, web traffic, social media, video streams and other sources, represents a huge challenge. The main problem is that the big data system is based on Hadoop technology, especially MapReduce for processing. This latter is a high scalability and fault tolerant framework. It also processes a large amount of data in batches and provides perception blast insight of older data, but it can only process a limited set of data. MapReduce is not appropriate for real time stream processing, and is very important to process data the moment they arrive at a fast response and a good decision making. Ergo the need for a new architecture that allows real-time data processing with high speed along with low latency. The major aim of the paper at hand is to give a clear survey of the different open sources technologies that exist for real-time data stream processing including their system architectures. We shall also provide a brand new architecture which is mainly based on previous comparisons of real-time processing powered with machine learning and storm technology.**

Keywords*—*Data stream processing; real-time processing; Apache Hadoop; Apache spark; Apache storm; Lambda architecture; Kappa architecture

I. INTRODUCTION

With the exponential growth of the interconnected world to the internet, a very large amount of data is produced coming in a form of continuous streams from several sources such as sensor networks, search engines, e-mail clients, social networks, e-commerce, computer logs, etc. McKinsey [1] relates that 5 billion individuals use diverse mobile devices. The increase of generating data doesn't have a limit. This phenomenon is known as "big data". According to the study of IBM on Big data, by 2020 there will be approximately 35 zetta bytes of data generated annually [2] and the data growth will be as high as 50 times than it is nowadays [3]. 2.5 quintillion bytes of data are produced every day [4]. Besides, in every second data are produced continuously; 34,722 Likes in Facebook, about 571 new websites, and almost 175 million tweets. All these multiple internet technology actors generate a very large data and information in the form of streams. These data are incremented in real-time according to the 5Vs of Gartner.

With recent technologies, such as applications of the Internet of Things, the stream of data is multiplied in volume, velocity and complexity. In addition to that these streams of data are processed with high velocity in real time which brings some unique challenges. The system of big data depends on the Hadoop framework which is considered as the most effective processing technique.

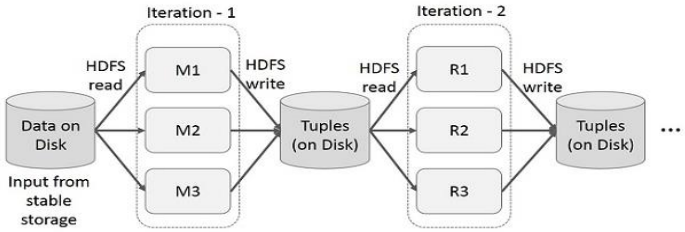

Fig. 1. Map reduce jobs.

Hadoop has a proportional performance to the complexity of large data. It is an effective tool for solving massive data problems. Despite the success it has had, this model has some limitations. Among the major limitations, Hadoop, precisely Map Reduce framework, is not the best tool for processing the latest version of data; it is limited to process data in batch mode. In other words, it cannot handle what is happening in real time. In several cases, it is very important to process data as created and have knowledge of what is happening in real time.

MapReduce is a simple programming model that permits processing a fixed amount of data but is not appropriate for real time stream processing. It is a batch processing system [5] which means that when the first batch is terminated, the data, that are offered for the final user, are aged at least until the second batch is terminated. Furthermore, MapReduce is appropriate for parallelizing processing on a big amount of data. However, it relies on a disk based approach. That is to say, each iteration output is written to disk making it slow. The following illustration in Fig. 1 shows that MapReduce reads the data from the disk and writes them back to the disk four times. There is one more disk read/write operation. In every MapReduce job, the disk reads and writes twice which makes the complete stream very slow and downgrades the performance. Hence, the need to create a tool that processes data immediately and gets a response of a query in real time with low latency.

In this paper we present an overview of some fundamental notions of big data, stream processing and the increasing volume of data. Then, we describe different tools and systems that permit processing data in real time. A thorough comparison is also taken into account in the following section. In addition, we propose a new architecture based on

the previous comparison. And last but not least, we compare our proposed architecture with the two existing architectures.

II. Big Data

Nowadays the term "big data" is frequently used in industry, science and so many other fields. Big data is a broad concept which refers to the explosion of large data sets that traditional database cannot process and manage due to the high volume and complexity of data generated in any time.

O'Reilly describes Big Data as [6] "data that exceeds the processing capacity of conventional database systems. The data is too big, moves too fast, or does not fit the structures of existing database architectures. To gain value from these data, there must be an alternative way to process it". According to McKinsey Global Institute [1], "Big Data" refers to datasets whose size is beyond the ability of typical database software tools to capture, store, manage and analyze. IDC's definition of Big Data technologies describes a new generation of technologies and architectures designed to economically extract value from very large volumes of a wide variety of data by enabling high-velocity capture, discovery, and/or analysis [7]. According to these definitions, big data refers to technologies that allow companies to quickly analyze a very large volume of data and get a synoptic view by mixing storage, integration, predictive analysis and applications. The Big data allows gaining time, efficiency and quality in data interpretation.

Gartner sees that this concept uses a set of tools and technologies to solve the problematic of the five Vs of Big data which include: Volume, Variety, Velocity, Veracity and Value. Volume stands for the very large quantity of data gathered by an organization, from datasets with sizes of terabytes to zetta bytes and beyond. Data comes from everywhere and the size continues to increase; by 2020 data will have become 44 times bigger (about 40 ZB) than that of 2009 [8]. Data come from a variety of sources and in many types. This is the second aspect of big data 'variety' [9] which refers to the various data types including structured, unstructured, or semi-structured data such as textual database, streaming data, sensor data, images, audios, videos, log files and more. These various types of data are going to be combined and analyzed together for producing a new insight. Velocity represents the speed of processing, analyzing and visualizing data for immediate response. Veracity stands for the reliability of data; the user must have confidence in the data to use for the right decision making. This characteristic is very hard to achieve with big data and represents an important challenge. Last but not least, Value which indicates the insights we can reveal within the data. It is not necessary to analyze and process a large data without value; rather we must focus on data with real value.

To summarize, big data do not refer to a huge volume of data and complex analysis, but on how to process, analyze and store rapidly at the moment this great size of information coming from different sources and in the shape of streams. This is done in order to obtain the right answer and make the best decision.

Unlike the traditional data processing systems, stream processing systems can process an unbounded source of events. It can also transform the streams of data in real time with low latency so as to get real time response and make processed data directly accessible for the final user. SQL Stream defines "stream processing [as] the real-time processing of data continuously, concurrently, and in a record-by-record fashion. It treats data not as static tables or files, but as a continuous infinite stream of data integrated from both live and historical sources".

Stream processing systems are invented to deal with big data in real time with a high scalability, high availability, and high fault tolerance architecture [10]. It permits to process data in motion as it is produced. We can say that a stream processing is a real time processing of continuous series of data stream by implementing a series of operations on every data point. The objectives of Stream processing are to collect, integrate, process, analyze and visualize the data as they arrive in real time to extract a greater insight. Processing streams of data in real time brings some challenges.

In the following section, we will give an overview of some tools of data stream processing and compare them so as to choose the best tool that satisfies the constraint of real time.

III. Data Processing Tools

In this section, we are going to present an overview of data stream processing tools including; Apache Hadoop, Apache spark and Apache storm to better understand very the difference between systems. Based on this description, we will show that older methods, precisely MapReduce, do not allow the processing in real time as it has the possibility to process a very large volume of data regardless of the speed with which the data arrives.

A. Apache Hadoop

The Apache Hadoop is a project of apache foundation, created by Doug Cutting in 2009, which is an open source software framework designed for scalable, reliable, and distributed computing. This framework allows the distributed processing of big data sets on clusters of computers. The Hadoop Framework includes several modules: Hadoop common, Hadoop Distributed files system (HDFS), Hadoop yarn, and Hadoop MapReduce. Fig. 2 represents the Hadoop ecosystem and shows the core components of this framework, namely, HDFS for storing a large volume of data sets and MapReduce for processing big data in batch mode.

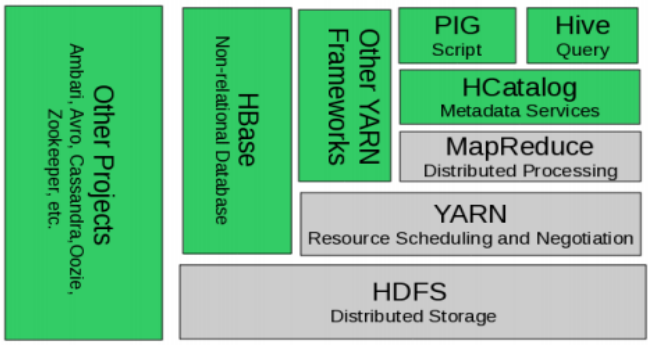

Fig. 2. Hadoop Ecosystem [11].

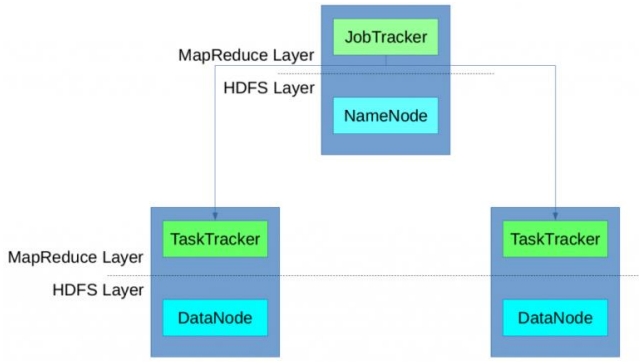

Fig. 3. Architecture of Hadoop [12].

Hadoop has a master/slave architecture that consists of two servers which are the basics of MapReduce framework, a single master node and several worker nodes [13]. A master node namely Job Tracker is responsible of accepting jobs from customers, dividing jobs into tasks, assigning tasks to worker nodes and re-executing failed tasks. Every worker executes a task tracker process which is responsible to execute and manage the tasks assigned by Job Tracker on a single computation node in the cluster as shown in Fig. 3. HDFS [14] is designed to be a distributed, scalable and resilient storage system that is designed to interact easily with MapReduce. It provides an important aggregation bandwidth throughout the network. HDFS is composed of a master node called Namenode and data servers called Datanodes. The structure of the HDFS file is divided into blocks of 128 MB.

MapReduce, which was first developed in 2004 by Google, is a framework whose role is to facilitate processing vast amount of data in parallel on large clusters of commodity hardware in a fault tolerant manner [12]. It is divided into two separate steps, namely, map phase and reduce phase [15], [16]. First, the user defines a map function to process the input data and produce a group of intermediate key/value pairs. Second, the intermediate values with the same intermediate key are grouped together by MapReduce library and transferred to the reduce function. And finally, the reduce function processes the intermediate results and finishes the job. Fig. 4 indicates the execution workflow of MapReduce job. The MapReduce library splits the input data into M disjunctive partitions for the parallel execution of map operation about 16-64 MB per piece [16]. The copies of program are launched on computer of the cluster. The Master assigns map and reduce tasks to running worker instance, the worker with map task reads assigned partition, processes all input pairs with map function, buffers output pairs in local main memory and flushes buffer periodically to disk. Storage location is reported to the master, which coordinates hand-over to reducers. Worker with reduce task gets the location of intermediate results and reads them. Shuffle means sorting pairs by key to group them and write results of reduce function into the output file that is associated with reducer's input partition. After all map and reduce task have been processed completely the master returns to wake up the user program [16].

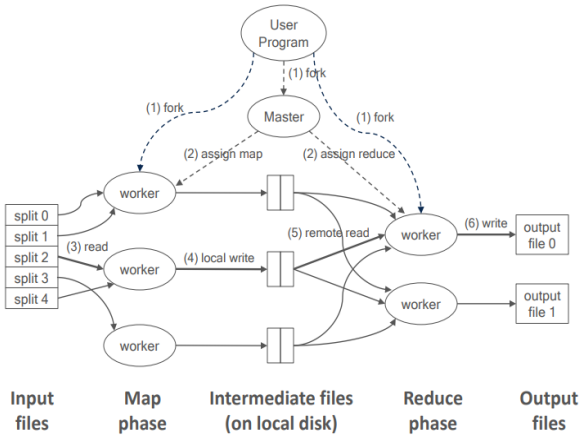

Fig. 4. Map reduce model.

MapReduce is a fault tolerant framework that processes a big amount of data due to its elasticity and scalability, but it is not a perfect way for real time data processing. The MapReduce programming model has some limitations. These limitations are presented as follows:

1) Only suitable for processing data on batch
2) No real time
3) Stock data on disk which makes Disk intensive
4) No repetitive queries
5) Not efficient for caching; MapReduce can't maintain the intermediate results in memory
6) Not efficient for iterative algorithms and interactive data querying
7) One-input and two-stage data flow is extremely rigid
8) Common operations must be coded at hand
9) Semantics hidden inside map reduce functions, difficult to maintain, extend and optimize

B. Apache Spark

Spark is an open source framework for distributed computing [17]. It is a set of tools and software components structured according to a defined architecture. It is developed and designed at the University of California at Berkeley by AMPLab. Spark is now a project of the Apache Foundation. This product, which is an application framework of big data processing to Spark, performs a data read at the cluster level (cluster of servers on a network), and performs all necessary analysis operations by writing the results at this same level. Despite the fact that it is written with Scala, Java and Python languages, it makes the best use of its capabilities with its native language 'Scala'.

The main difference between MapReduce and spark is that MapReduce from Hadoop works on stages while Spark works on all the data at the same time. It is up to ten times faster for batch processing and up to a hundred times faster for performing in-memory analysis. Spark performs all the data analysis operations in memory and in real time [18]. It relies on disks only when its memory is no longer sufficient. Conversely, with Hadoop the data are written to disk after each operation. This work in memory reduces latency between treatments which explains such rapidity.

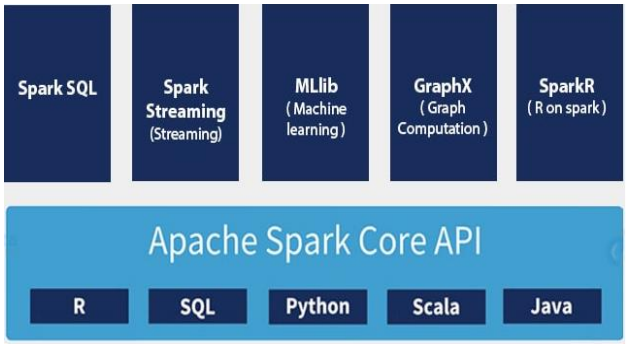

Fig. 5. Spark framework ecosystem.

Fig. 6. Spark streaming.

However, Spark does not have a file management system of its own. It is necessary to provide one, e.g. Hadoop Distributed File System, Informix, Cassandra, OpenStack Swift or Amazon. It is recommended to use it with Hadoop which is currently the best overall storage solution thanks to its more advanced administration, security and monitoring tools. In case of failure or system failure: Data objects are stored in so-called resilient distributed datasets (RDDs) distributed over the data cluster for a complete data recovery.

Spark wants to be a response to the limitations of MapReduce and allows in the same environment to easily access a wide variety of use cases, such as SQL, Streaming, Machine Learning and Graph Analysis, in a more efficient and interactive way as shown in Fig. 5.

Spark Streaming is a programming interface for processing data flows on a Spark platform. Data can be received in a variety of ways: file system transfers, TCP sockets reception (generic network connections), or Twitter, Kafka, Flume, etc. Fig. 6 [19] shows several operations that can be directly applied to the streams. Each stream is being represented by a DStream, It is a transformation of a stream to obtain another stream, merging of several streams into one, joining of stream, joining between a stream and a single RDD, filtering a stream from another stream, updating a state from a stream, and so on. These operations can be applied both through spark-shell and from a program.

C. Apache Storm

Storm [20] is a real-time computing system that is distributed, fault-tolerant and guarantees data processing. Storm was created at BackType which is a company acquired by Twitter in 2011. It is an open source and open source project under the Eclipse Public License. The EPL is a very permissive license, allowing you to use Storm either in open source or for proprietary purposes. Storm makes the processing of unlimited data flows clear and reliable, making for real-time processing what Hadoop has done for batch processing. Storm is very simple and has been designed from the ground up to be usable with any programming language.

Storm can be used for some different use cases:

- Streams/flows processing: Storm can be used to process a stream of new data and update databases in real time.
- Continuous calculation: Storm can make a continuous query and disseminate the results to customers in real time.
- Distributed RPC: Storm can be applied to parallelize an intense request on the fly. If Storm is configured correctly, it can also be very fast: a frame rate reference of more than one million tuples treated per second per node.

A storm cluster consists of three nodes: “Nimbus” which is equivalent to the Hadoop Job Tracker, “Supervisor” that is responsible for initiating and terminating the process, and “Zookeeper” node which is a shared coordination service that directs the storm cluster as explained in Fig. 7.

Instead of using “MapReduce jobs” like in Hadoop, we use “topologies” in Apache Storm. It consists of spouts and bolts with bonds among them to show how streams are passing encompassing. We describe it as a data processing Directed Acyclic Graph (DAG) which draws the entire stream processing method. A topology design is presented below in Fig. 8.

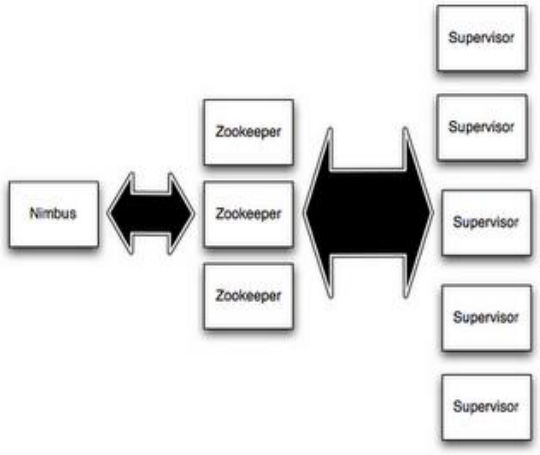

Fig. 7. Storm architecture.

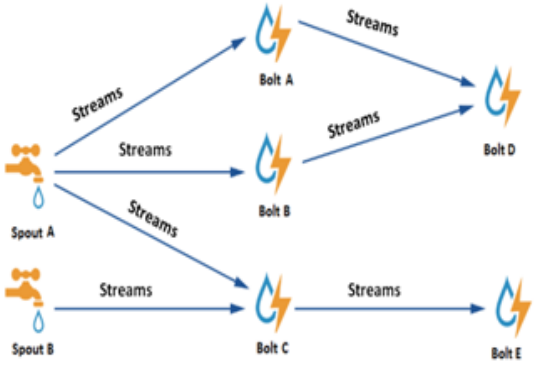

Fig. 8. Topology of storm.

IV. A Comparison of Data Processing Technologies

In this section, we differentiate between the different tools used for the real-time stream processing and based on this comparison; we will determine the most suitable tool.

TABLE I. Data Processing Technologies

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

The illustration above in Table 1 shows that storm is the best tool for real-time stream processing, Hadoop performs batch processing, and spark is able of doing micro-batching. Storm employs the spouts and bolts to do one-at-a-time processing to avoid the inherent latency overhead inflicted by batching and micro-batching.

V. Real Time Processing Architectures

In this part, we will present two architectures based on real-time processing called Lambda and Kappa. According to this description, we will compare them then deduce a more robust architecture that satisfies the real-time constraint.

A. Lambda Architecture

The lambda architecture unifies real-time and batch processing in a single framework which provides low latency and better results. It was founded thanks to Nathan Marz's motivation to build the hybrid system.

The lambda architecture [21], shown in Fig. 9, consists of three layers and each of these layers can be made using various large technologies, described as follows:

Batch layer: Stores the master copy of dataset and computes arbitrary batch views.

Serving layer: Integrates results from the batch and speed layer.

Speed layer: Only processes the recent data to compensate the high latency of the services layer updates.

Firstly, all the original data streams are dispatched to the batch and speed layer for processing. **The Batch layer** allows batch processing for pre-computation of large amounts of datasets. It provides the managing of the Master Dataset; a set of immutable, append-only and exclusive raw data, but also provides a pre-computation of arbitrary query functions, called batch views. This layer doesn't update regularly batch views which lead to latency. MapReduce is a good example of batch processing that can be used at the level of this layer. Secondly, **the Serving layer** means computing in Real-time (Speed time) to minimize latency by performing real-time calculations as data arrive. This layer indexes batch views produced by the batch layer so that they can be queried in Ad-Hoc with low latency. Typically, technologies such as HBase, Impala, and Cassandra can be used to implement this layer. And finally, **Speed layer** which responses to queries, interfacing, querying and providing calculation results. This layer accepts all requests that are subject to low latency requirements, using fast and incremental algorithms but only deals with recent data. In this layer, we can use stream processing technologies like Apache spark, SQLstream, Apache storm. In a high-level point of view, the figure below shows the basic architecture and how the Lambda architecture works.

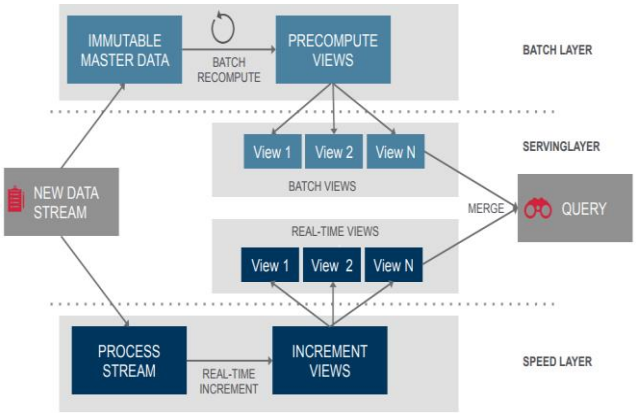

Fig. 9. Lambda generic architecture [22].

The lambda architecture has some flaws [23]:

- The business logic is implemented twice in the real-time and batch layers. The developers need to maintain code in two separate distributed systems.
- Lambda is an architecture for asynchronous processing. Hence, the computed results are not immediately consistent with the incoming data.
- Resulting operational complexity of systems implementing the Lambda architecture is huge.

- The operational burden of managing and tuning two operating systems for batch and speed layers is very high.
- Need for more frameworks to master.
- More straightforward solutions when the need is less complicated.

B. Kappa Architecture

Kappa architecture [24] is a simplification of lambda architecture. It was created by Jay Kreps in 2014 by the experience in LinkedIn and is a software architecture pattern. A Kappa architecture system is like the lambda architecture with the batch processing system eliminated. To replace batch processing, data is transmitted merely through the streaming system rapidly [24]. Rather than utilizing a relational DB like SQL or a key-value store similar to Cassandra, the canonical data store in a Kappa Architecture system is an append-only permanent log. From the log, data is streamed to a computational system and forwarded into auxiliary stores for serving.

Unlike the Lambda architecture, the Kappa architecture is more dedicated to processing data. It does not allow their permanent storage. Even though it is limited, the Kappa employs only a single code path for the two layers which reduces system complexity [25] as opposed to lambda architecture, which uses two separate code routes for the batch and the speed layer. The Kappa architecture illustrated in figure 10 is composed of two layers: The stream processing layer which executes the stream processing jobs and the serving layer which is used to query the results.

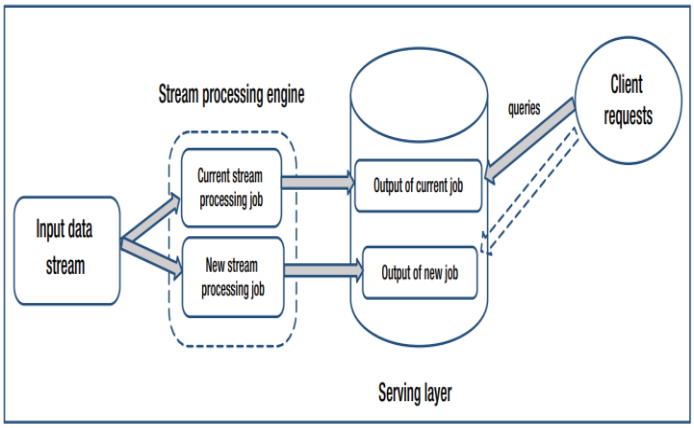

Fig. 10. Kappa architecture [23].

The advantages of Kappa architecture is allowing users to develop, test, debug and operate their systems on top of a particular processing framework. The Kappa architecture can be implemented using various technologies like Apache Storm, Spark, Kafka, HBase, HDFS or Samza. This architecture has been chosen to meet the need for data consistency and streaming processing because it allows a real-time and reliable execution of its log system.

Table 2 presents a short comparison of the two architectures as has been explained before, specifically Lambda and Kappa, following particular criteria.

TABLE II. A COMPARISON OF LAMBDA AND KAPPA ARCHITECTURES

Architectures / Criteria	Lambda architecture	Kappa architecture
Architecture	Immutable	Immutable
Layers	Batch, serving and real-time layer	Stream processing and serving layer
Processing data	Batch and real-time	real-time
Processing guarantees	Yes in batch but approximate in streaming	Exactly once with consistency
Re-processing paradigm	In every batch cycle	Just when code change
Scalability	Yes	Yes
Fault tolerance	Yes	Yes
permanent storage	Yes	No
Real-time	Isn't accurate	Accurate

VI. PROPOSED ARCHITECTURE

Many advantages and drawbacks of the two architectures were presented. Based on what has been noted in previous paragraphs, we have designed a novel architecture that is open source and follows a different set of characteristics mainly its ability to process large data in real time at high speed. In addition to that, it allows a limitless number of users to set up several new and creative features as well as applying many reforms.

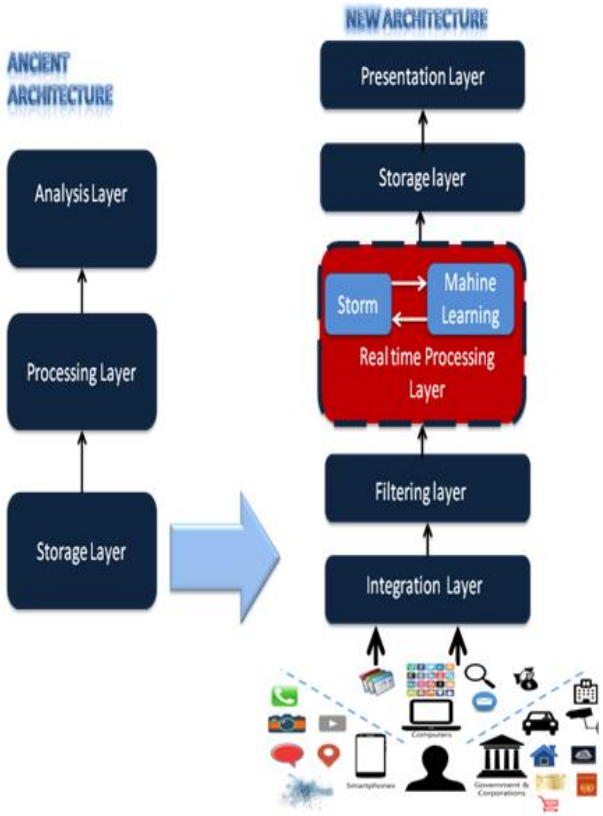

Fig. 11. Proposed architecture.

The architecture at hand has to gather- organize- integrate-process-analyze-store and visualizes influent data streams with low latency. Thus the responding of the system ought to be fast depending on the used architecture be it spark or storm, or the amount of the data and the complicatedness of the performed calculations. Nevertheless, the choice of the most suitable and efficient medium or tool should be taken into consideration as it has to be relatively easy to use allowing the analysts or the developers to deal with infrastructure problems.

Ideally, we aim to create an architecture that permits to make a transition to scale uncomplicated and visually changing resource allocation. Moreover, the configured resources must be chained to the cluster and should deal with changes in load or traffic without interruption. Finally, this architecture has to offer a live visualization of streaming data. It should also allow the creation of dashboards, custom graphics as well as UI extensions.

Both traditional architectures of big data and the proposed one are represented in Fig. 11. The traditional architecture consists of three layers viz. storage, processing, and analysis. On the other hand, our newly proposed architecture works differently. That is to say, the data incoming as a stream from various sources, like social media, cyber-infrastructure, web, sensors, email, and networks, come with a high speed. These data are delivered on time as they occur in the integration layer. This latter acquires the use of a set of tools and functionalities as is the case of Apache Kafka.

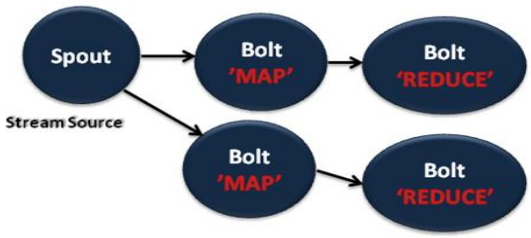

Fig. 12. Real-time processing Layer.

This layer makes it possible for the data to be ingested whatever are the formats and velocity. The data are going to be filtered into ELT, extract-transform-load operations (e.g., PIG), directly after being ingested. This layer is an important phase to filter streams of data in real time data processing. That is to say, the data will be cleaned and their qualities analyzed. This preprocessing stage, the filtering layer, gets rid of unwanted fields and special characters to make the processing and analysis reliable. To filter data streams we are going to use some algorithms such as sliding window, load shedding and synopsis data model. All this leads to the preparation of data for the real-time processing layer which mainly targets the processing of data in real time and with reduced latency. In this layer, we need robust and dynamic algorithms to confront the diversity of data. Fig. 11 represents two technologies that are used in this layer specifically storm and machine learning. The use of this latter in the present layer permits to archive the data and its objective is to visualize recent trends through a request/respond tool on similar inputs. It learns continuously from the newly arriving data the thing that makes the processing easy. On the other hand, in this layer storm is used to process the data in real time as it uses the so-called topology which is a network of Spout and Bolt. As mentioned previously, the streams arrive from Spout which broadcasts data arriving from external sources in Storm topology.

In Bolts, many functionalities can be used including filters, functions, joins, aggregations, etc. Consequently, we can apply map function in Bolt to mark the words of the stream. This resulting stream which comes from Bolt 'Map' proceeds into the following Bolt which implements the 'Reduce' function to aggregate the words into numbers as shows Fig. 12.

After the processing phase meets an end, the storage layer takes over. The storage is performed at the level of HBase. After the database is prepared and configured, region servers are created, and finally, backup and tables are mastered. The main role of the visualization layer is to present to the user the final data and results in streaming mode. This layer can give a quick response if all phases are achieved successfully.

VII. Comparison with Related Architectures

The proposed architecture has been put in place to deal with some problems at the level of both lambda and Kappa architectures. Lambda allows providing customers with the freshest vision possible. However the business logic is implemented at the level of both layers, two different sources of the same data are needed namely files and Web Services, and several frameworks are necessary to set up this architecture. Consequently, Kappa architecture was born in response to the complexity of lambda architecture. Unlike lambda, Kappa brings an evolution in a way that it is more dedicated to data processing even though it does not permit the permanent storage of data. This architecture is more straightforward than lambda and gives the user the freedom to single out the composers of implementation. Nevertheless, Kappa does not have a separation between the needs and is not a magical spell to solve all the problems in big data. In addition to that, these two architectures focus on addressing performance issues by balancing throughput and latency rather than data quality issues and data analysis results.

Our architecture, on the other hand, is based on the principle of Kappa architecture. It is a data processing streaming approach that processes all incoming data as streaming data and allows permanent data storage. It can also provide real-time processing by using storm and machine learning. Storm, which is a distributed real-time computing system, is fault tolerant as it manages the errors happening in working procedure and nodes. This method does for real-time processing what Hadoop does for batch processing. Storm can quickly compile and expand complicated real-time computation in a computer cluster and permit to process endless streams of data reliably.

Topologies of storm should be created inside it to realize real-time computation. In this layer, we integrate a distributed machine learning algorithms. Traditional supervised machine learning algorithms form data models based on historical and static data, whereas Traditional unsupervised machine learning re-examines all datasets if new data analysis is needed to detect the pattern.

Conversely, our architecture is going to use both supervised and unsupervised approaches to implement the distributed streaming version of adopted machine learning algorithms. The supervised learning in streaming approach learns continuously as new data arrives and is labeled. Unlike the traditional one, unsupervised learning in streaming approach can detect unusual patterns in streaming data in real time without any reexamination of the data that were analyzed before.

VIII. Conclusion

With the recent evolution of big data, processing a large amount of data becomes a big challenge. Map-reduce technology provides a distributed computing platform for processing a large dataset on larger clusters. Nevertheless, it does not satisfy the real-time processing capacity, hence the need for a strong system that meets these expectations to overcome the limitations of the traditional system.

The main goal of this paper is to propose a real-time processing architecture that builds on the storm technology as well as machine learning. Storm allows processing a very large volume of data with low latency and high velocity. Machine learning, on the other hand, learns continuously from new coming data which facilitates processing. Furthermore, this proposed architecture is based on a survey of open-source real-time processing systems, including Hadoop, spark, and storm. Two major architectures, namely lambda and kappa, were compared to create a brand new strong one.

In this proposed architecture, we suggested giving priority to real-time processing layer, and we tried our best to enhance it by integrating storm and machine learning. This new architecture was mainly inspired by the advantages of lambda and kappa. Our next step is to validate and evaluate its performance. We also decided to analyze the same dataset with several machine learning techniques. In addition to that, we intend to build a real-time stream processing framework for IoT and sensing environment. Most studies are constrained by a few limitations, and this research is no exception. However, we cannot talk about its limitations until the validation stage is finished.

References

[1] "Big data : The next frontier for innovation, competition, and productivity," no. June 2011.

[2] "Driving marketing effectiveness by managing the blood of big data," IBM Corp., 2012.

[3] F. Pivec, The global information technology report 2003–2004, vol. 8, no. 4. 2003.

[4] BSA, "What's the Big Deal With Data?," Bsa, 2015.

[5] T. White, Hadoop: The definitive guide, vol. 54. 2015.

[6] E. Dumbill, "What is big data? - O'Reilly Media." pp. 1–9, 2012.

[7] P. Carter, "Big Data Analytics: Future Architectures , Skills and Roadmaps for the CIO," IDC White Pap., no. September 2011, p. 14, 2011.

[8] N. Khan et al., "Big Data: Survey, Technologies, Opportunities, and Challenges," Sci. World J., vol. 2014, pp. 1–18, 2014.

[9] H. J. Hadi, A. H. Shnain, S. Hadishaheed, and A. H. Ahmad, "Big Data and Five V ' S Characteristics," no. November, pp. 29–36, 2014.

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

[11] A. C. Murphy and V. K. Vavilapalli, "Apache Hadoop YARN," p. 2015, 2009.

[12] The Apache Software Foundation, "Apache Hadoop," 2007.

[13] M. de Kruijf and K. Sankaralingam, "MapReduce Online," IBM J. Res. Dev., vol. 53, no. 5, p. 10:1-10:12, 2009.

[14] A. Hadoop and C. Spotlight, "Apache Hadoop * Community Spotlight Apache * HDFS *," no. October, 2012.

[15] T. Chen, D. Dai, Y. Huang, and X. Zhou, "MapReduce On Stream Processing," pp. 337–341.

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

[17] S. Salloum, R. Dautov, X. Chen, P. X. Peng, and J. Z. Huang, "Big data analytics on Apache Spark," Int. J. Data Sci. Anal., vol. 1, no. 3–4, pp. 145–164, 2016.

[18] I. Ganelin, E. Orhian, K. Sasaki, and B. York, Spark: Big Data Cluster Computing in Production. 2016.

[19] M. Zaharia, T. Das, H. Li, T. Hunter, S. Shenker, and I. Stoica, "Discretized Streams: Fault-Tolerant Streaming Computation at Scale," Sosp, no. 1, pp. 423–438, 2013.

[20] S. Landset, T. M. Khoshgoftaar, A. N. Richter, and T. Hasanin, "A survey of open source tools for machine learning with big data in the Hadoop ecosystem," J. Big Data, vol. 2, no. 1, p. 24, 2015.

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

[22] "Lambda Architecture » λ lambda-architecture.net." 2014.

[23] B. Lakhe, Practical Hadoop Migration. 2016.

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

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