International Journal of Science and Research (IJSR)
ISSN: 2319-7064
SJIF (2019): 7.583

Real-Time Stream Processing of Big Data

Denis Patrick Bell[1], Eliasu Tambominyi[2], Yang Chunting[3]

[1]School of Information and Electronic Engineering, Zhejiang University of Science and Technology, No.318, Liu-he-Road, Hangzhou, Zhejiang Province, 310023, China
911808802009[at]zust.edu.cn

[2]School of Automation and Electrical Engineering, Zhejiang University of Science and Technology, No.318, Liu-he-Road, Hangzhou, Zhejiang Province, 310023, China
911807802002[at]zust.edu.cn

[3]School of Information and Electronic Engineering, Zhejiang University of Science and Technology, No.318, Liu-he-Road, Hangzhou, Zhejiang Province, 310023, China
yangct[at]zust.edu.cn

Abstract: ***The evolution of technology has enabled the continuous generation of massive data from connected devices and sensors. As, more data becomes available, organizations are using cutting-edge tools and techniques to extract useful insight from the data immediately they are generated. Therefore, the enterprise systems are evolving and shifting towards real-time systems. So, there is a growing need for well-developed fault tolerant distributed scalable systems to handle this change with low latency. Recent years have seen the emergence of several distributed systems whose popularity is as a result of the growing demand to efficiently analyze and interpret voluminous data. This article will discuss distributed real-time stream processing of big data, the two main real-time big data processing architectures: Lambda and Kappa and the popular frameworks used for processing real time big data.***

Keywords: Real-time data processing, Big data, Batch processing, Lambda & Kappa architecture

1. Introduction

Real Time data streaming though not a new phenomenon has seen considerable advancement in recent years. Progress in technological development and implementation coupled with increasing connectivity between human and communication devices, through the internet in most cases is constantly presenting the world with an insurmountable proportion of data. This ever growing sea of data is the reason for the growing demand and use of real time stream processing.

Streaming data is often generated from multiple sources continuously. For instance, data streams from sensors can be produced several times in just a few second. A streaming system need to constantly handle this data as they are received. Big Data is often used to describe extremely large volume of structured or unstructured data which are generated in terabytes, petabytes, exabytes etc. This includes data generated at high speed by sensors, web and mobile data from social networks, e-commerce etc.

Big data and its challenges can be easily characterized by five Vs: Volume, Velocity, Veracity, Variety, and Value [1].Volume here represents the amount of data. The amount of data stored is in many terabytes, petabytes and may soon reachzeta-bytes in the nearby future. Velocity denotes the actual speed of data processing. Data from sensors may need to be process in few milliseconds in real time. Veracity is the accuracy of the data. Variety is different types of data that exist such as images, audio, video, text and so on. Data is produced from various sources in different forms as structured, semi structured or unstructured.

Data form an integral part of any enterprise. When size of data collected is increased, the task of processing the data for business intelligence becomes more challenging making traditional techniques computationally expensive. Raw data is useless unless processed to extract useful information. Data can be process using basically two techniques: batch processing and streaming (also called stream processing). Batch processing involves grouping large volume of data collected for a transaction over a period of time, then processed to obtain batch results. Streaming on the other hand is done in real time under streaming process [1].

The shift towards more dynamic and user-generated content in the web and the unstoppable emergence of smart devices possessing modern communication utilities and other mobile devices, in particular, have led to an abundance of information that are only valuable for a short time and therefore have to be processed immediately[7]. This ever growing widespread use of smart devices has allowed the end-user to share experiences online about services or products almost immediately through either purchasing sites or popular social networking sites which is figured by companies as valuable and realistic. This has resulted in many corporations and companies such Amazon, Taobao, Alibaba, Netflix, Facebook, New York Stock Exchange etc. to be already adapted at monitoring user activity for the purpose of improving services. Twitter performs continuous sentiment analysis to inform users on trending topics as they come up and even Google has parted with batch processing for indexing the web to minimise the latency by which it reflects new and updated sites [5]. We can highlight two major tasks of data processing [4] stream which are processing queries of data stream and data mining.

Data mining streams enables us to extract knowledge from continuous data streams. Several technologies have emerged

specifically to address the challenges of processing high-volume, real-time data [2].

The rest of this paper is organized as follows. In Section II, we introduce stream processing. In section III, we present some cutting-edge support tools for big data stream processing. In section IV the discussion is based on the Lambda and Kappa architecture. Section V and IV are discussion and conclusion respectively.

2. Stream Processing

Massive data continuously generated from multiple sources simultaneously in the form of data records is called Streaming data. This data is obtained from a variety of sources such as sensor networks, web or mobile application click logs, Internet of Things (IoT) objects, tags (beacons), machine to machine communications (M2M) etc.[4]. The data requires processing on a record-by-record basis to draw valuable information [1].

Real-time analysis enables us to process data as they are generated, therefore, we can get useful insights on the data in motion. Stream processing is extremely useful in situation where dynamic new data is generated continuously. Nevertheless, numerous applications like environmental monitoring and fraud detection applications require continuous and timely processing of information [2].

There are many use cases of data streaming in real-life some of which are:

- Streams to provide relevant recommendations in web applications in real-time thereby improving user experience
- Stock price movements are tracked in the real time to evaluate risks and portfolios are automatically balanced [1]
- Banks collect real-time data to monitor transactions and trigger alerts in case of fraud
- Real-time responsiveness to changing market conditions [4]
- Streaming to provide trending topics on social media platforms
- View recent modification on a website in real-time.
- Gaming platform used streaming to collect real-time data for game-player communication and this enable the players to take action.
- Monitor traffic streaming on a network [4]

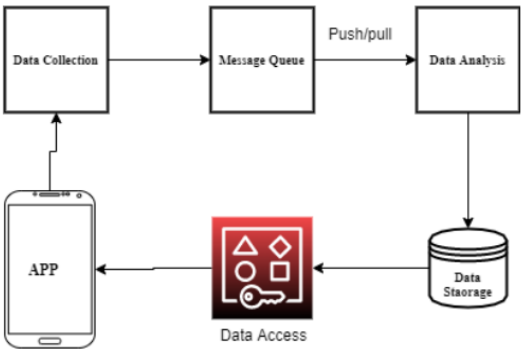

Figure 1: Real-time Application Implementation

3. Real-Time Stream Processing Framework

Popular real time streaming computation tools include:

3.1 Apache Spark

Apache Spark is an open-source unified analytics engine for large-scale data processing that supports in-memory processing to boost the performance of applications that analyse big data[3]. Four years after it's inception as a UC Berkeley project in 2009, it became a member of the Apache Software Foundation in 2013 subsequently becoming a top-level project. The current version of spark at the time of this paper in the 2.x line is version 2.4.7 released on 12th September 2020 while in the 3.x line, version 3.0.1 was released on 8th September 2020.

Apache spark achieves high performance for both batch and streaming data using a state of the art DAG (Directed Acyclic Graph) scheduler, a query optimizer and a physical execution engine [3]. Spark is the most active project in terms of community numbers and utilizes the micro batch technique [2]. Spark streaming is API used for handling streaming data. A Spark deployment consists of a cluster manager for resource management, a driver program for application scheduling and several worker nodes to execute the application logic [7].

Spark Streaming [17] is an extension of the core Spark API that enables scalable, high-throughput, fault-tolerant stream processing of live data streams. Data can be ingested from many sources like Kafka, Flume, Twitter, ZeroMQ, Kinesis, or TCP sockets, and can be processed using complex algorithms expressed with high-level functions like map, reduce, join and window (figure 2)].

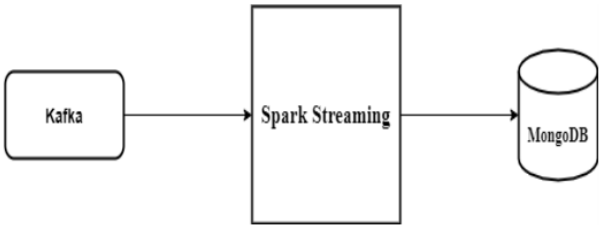

Figure2: Spark Streaming

Finally, processed data can be pushed out to files systems, databases, and live dashboards. In fact, you can apply Spark's machine learning and graph processing algorithms on data stream [4].

Apache Spark improves upon the Apache Hadoop framework (Apache Software Foundation, 2011) for distributed computing, and was later extended with streaming support. To implement in-memory computations, Spark uses a model called Resilient Distributed Datasets (RDDs), its in-memory abstraction to work with data[17]. It also consists of Spark SQL, Spark Streaming, machine library and GraphX. The Spark Core is the foundation responsible for basic input/output functions, task dispatching and scheduling. Spark runs on top of Mesos, YARN or in standalone mode in which case it may be used in combination with Zoo Keeper to remove the master node[7]. Spark Streaming shifts Spark's batch-processing approach

towards real-time requirements by chunking the stream of incoming data items into small batches, transforming them into RDDs and processing them [8]. Spark is lighting fast as it runs on memory clusters. Itis proven to be ten times faster than Hadoop's disk-based Apache Mahout due to the distributed memory-based Spark architecture. It implements common algorithms to simplify large scale machine learning pipelines, like logistic or linear regression, decision trees or k-means clustering.

3.2 Apache Storm

Apache Storm is a free and open source distributed realtime computation system which makes it easy to reliably process unbounded streams of data, doing for realtime processing what Hadoop did for batch processing [8]. The concept of "stream" is used in apache storm to mean a distributed abstraction of data in motion. It adopts a fault tolerant and reliable model, which can be challenging to achieve with traditional queues and workers architecture.

The Storm cluster consists of master nodes and worker nodes. Master node runs a daemon called Nimbus, which is the central component of Apache Storm [11]. Nimbus responsibly analyzes, gather tasks and then execute thetopology. The Nimbus distributes task to the available worker node also called supervisor. The supervisor has one or more worker process and delegate tasks to the worker processes. Storm has an internal messaging system to facilitate communication between the master and the worker nodes. Storm is stateless and relies on Zookeeper framework to maintain state. The reliability of the Zookeeper enables the failed Nimbus to easily restart and continue the execution of the topology.

Early versions of Storm introduced the common stream abstraction and the corresponding Spout/Bolt/Topology API that allowed developers to easily reason about streaming computations. Storm now supports reliable state implementations that can withstand and recover from supervisor failure. Version 2.2.0 released on 30th Jun 2020 enables HB timeout configurable on a topology level, enables SSL credentials to autoload for storm user-interface, LogViewer and DPRC server [9]. This version also allows health check to pass on timeout.

3.3 Apache Samza

Apache Samza is a distributed stream processing framework which uses Apache Kafka for messaging, and Apache Hadoop YARN to provide fault tolerance, processor isolation, security, and resource management [2], [4]. Samza logically scales application by breaking it down into multiple units known as tasks which consume data from single partition of the input data[11].

Samza was co-developed with the queueing system Kafka, therefore relies on the same messaging semantics and Apache YARN for the distribution of tasks among nodes in a cluster [7], [2]. The main goals of Apache Samza are having better fault tolerance, processor isolation, security, and resource management [10].

The architecture of Samza is similar to that of Hadoop which uses HDFS (hadoop distributed file system) as storage layer, YARN for execution layer and MapReduce as the processing technique and programming model[11].

In comparison to Storm, Samza requires a little more work to deploy as it does not only depend on a Zookeeper cluster. It runs on top of Hadoop YARN for fault tolerance [7]. Samza allows tasks to maintain state by storing it on disk (typically using Kafka) [2].

Samza applications are made of streams and Jobs. Streams basically are composed of immutable sequence of messages of similar category. The Streams are partitioned and messages (i.e. data items) inside the same partition are ordered, whereas there is no order between messages of different partitions [7]. Each stream is partition to allow scalability of the system so that the system will be capable of handling large scale data. Input streams of tuples are decomposed and partitioned so that data flow graph is created[10]. Jobs consume and process the set of input stream. Jobs are then broken down into tasks which are smaller units of execution used to scale the throughput of jobs processor. Scalability is achieved through running a Samza job in several parallel tasks each of which consumes a separate Kafka partition; the degree of parallelism, i.e. the number of tasks, cannot be increased dynamically at runtime [7]. The messages are independent i.e. there is no particular order of messages across the partitions. Therefore, task will be executed independently.

3.4 Apache Flink

Apache Flinkis a framework and distributed stream process engine for stateful computations over unbounded and bounded data streams[13]. Flink is scalable, fault tolerant, performs computations at in-memory speed and can work for all common cluster environment. Apache Flink is a distributed system for processing both bounded and unbounded data. Bounded streams have a start and an end while unbounded has a start but doesn't have an end.

Flink integrates well with Hadoop, YARN, Kubernates, Apache Mesos but can also operate as stand-alone cluster. Flink provides true stream processing with batch processing support[2].

Flink has a kappa architecture. Kappa architecture is used for processing streaming data. Kappa architecture however is use for both real-time and batch processing through a single processing engine. Batch processing in kappa architecture is a special case of streaming. This type of architecture is helpful for analytics with a single technology stack.

Flink architecture is similar to master-slave architecture where the Job Manager acts as master and the Task Manager as slave. The Job Manager is used for coordination of all the computations in the Flink system, and the Task Managers are being used as workers and execute parts of the parallel programs [10].

Flink keeps track of distributed tasks, decides when to schedule the next task (or set of tasks), and reacts to finished tasks or execution failures[2]. Flink is famous for its ability to compute common operations such as hashing, very efficiently [10].

3.5 Other Real Time Stream Frameworks

Other frameworks worth noting include:

- **Amazon Kinesis**
  Amazon kinesis is a popular open-source flexible data streaming platform.
- **Apache Kudu**
  Apache Kudu is used by Wall Street to uncover fraudulent events and by Xiaomi (Cell Phone Company)to collect error reports.
- **Azure Stream Analytics**
  Azure Stream Analytics accommodates spontaneous data processing and can effectively accomplish tasks in a limited time period.
- **Apache Nifi**
  Apache Nifi automates the flow of data between the data point of origin and its destination. Nifi supports log files, social feeds e.t.c
- **Google Cloud Dataflow**
  This is an integration of cloud dataflow with python 3 and python sdk which works by removing meaningless data that can decrease the rate of analysis.

4. Architecture

The need to manage different workloads under a coherent architecture led to several design patterns with the most popular being the 'Lambda Architecture' [2].

The complexity of using different batch and streaming architectures paved the way for the 'Kappa architectural' pattern that fuses the batch and stream layers together [2].

4.1 Lambda Architecture

Lambda architecture (LA) is a data processing technique capable of processing massive quantities of data (i.e. Big data). It's considered as a near real time data processing architecture. Lambda architecture is an approach aimed [14] at satisfying the needs of a robust system that is fault-tolerant, both against hardware failures and human mistakes, being able to serve a wide range of workloads and use cases in which low-latency reads and updates are required.

The lambda architecture [12] is divided into three layers: batch layer, serving layer, and speed layer.

The batch layer manages the master dataset and pre-computes the batch views.

The serving layer is a data store that queries view from both the batch and speed layers.

The speed layer creates real time views for the end-user. (figure 3).

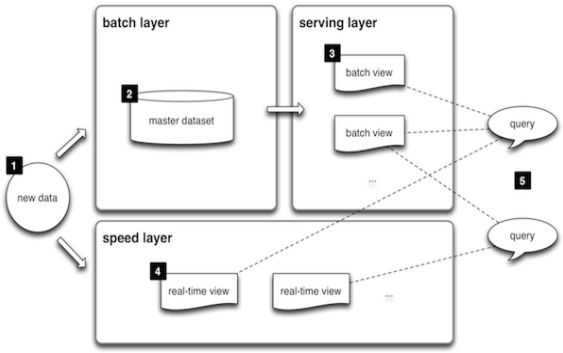

Figure 3: Lambda Architecture [12]

The way Lambda architecture works is that an immutable sequence of records is captured and fed into a batch system and a stream processing system in parallel[15]. The new data comes into the systems continuously and is dispatched to both the batch layer and the speed layer, then the output from the batch layer (batch views) and those coming from the speed layer (near real-time views) get forwarded to the serving layer which indexes the batch views so that they can be queried in low-latency, ad-hoc way.

The lambda architecture is a scalable architecture for data processing which leverages good balance between speed and reliability. Batch layer manages historical data which makes data recovery possible in case of system failure.

The lambda architecture too has some disadvantages. One of the obvious problems here is coding overhead in which the developer writes the same code twice one for batch and the other for real-time layers.

In software engineering, we seek to solve the problem of duplicating source code. Therefore, one proposed approach to fixing this is to have a language or framework that abstracts over both the real-time and batch framework[18].

4.2 Kappa Architecture

Kappa architecture is software architectural pattern which focus on stream processing. It's is derived from the Lambda architecture but not the replacement for Lambda architecture. The Kappa Architecture, in contrast to Lambda Architecture, allows for more flexible adaptation to changing processing and analytics requirements since the overhead of a second processing layer is mitigated [16]. In this architecture incoming data are streamed through the real-time layer and the result then enters the serving layer for queries. Providing increased flexibility and reduced overhead, the Kappa Architecture is not without compromise[7]. The main drawback of this architecture involves handling duplicate events and cross-referencing events.

For this architecture, incoming data is streamed through a real-time layer and the results of which are placed in the serving layer for queries. The idea is to handle both real-time data processing and continuous reprocessing in a single stream processing engine. This requires that the incoming data stream can be replayed (very quickly), either

in its entirety or from a specific position. If there are any code changes, then a second stream process would replay all previous data through the latest real-time engine and replace the data stored in the serving layer. The Kappa architecture is simplified by keeping a single code base unlike the Lambda architecture in which a code base is maintained for every batch and speed layer. The operations however are generally easier to do in batch processing.

5. Discussion

Table 1 shows a comparison of the characteristics of some of the streaming technologies comparing features like architecture, throughput, latency integration, fault tolerance and their availability. Since each technology has its draw-backs, it is fair to conclude that integration of some of these technologies can make up for their short-falls by complementing each other, there by augmenting outputs. Except Apache spark which has a high latency, all the others in the table have low latency; in some cases very low latency. This distinction places Apache spark at a disadvantage to the other technologies with respect to that characteristic. Similarly Apache nifi is at a disadvantage to the other technologies because of its lack of fault tolerance. A broader issue relating to these comparisons is that there is no single universal standard for evaluating distinctions between these technologies. Even though it may seem like the technologies listed in the table have almost all the same properties, the varying strengths of the properties for a particular technology distinguishes it from the others. As a result, integration with one and the other can compensate for their respective weakness. Effective streaming relyon several features such as a high throughput, low latency, in-memory storage for stateful operations and fault tolerance. Almost all the technologies listed in the table are open source but Apache spark has the largest developer community because it integrates well with hadoop, has a high throughput, high fault tolerance, integrate well with many databases and can be used in machine learning. Apache Samza integrates well with kafka taking advantages of the strong features of kafka to complement it's streaming capabilities.

Table 1: Comparison of real-time streaming platforms

PLATFORM	Architecture	Throughput	State Management	Integration	Fault Tolerant	Latency	Availability
APACHE SPARK	Master/Worker	High	Yes	Yes	Yes	High/few seconds	Open Source
APACHE STORM	Master/Worker	High	Yes	Yes	Yes	Low/sub second	Open Source
APACHE FLINK	Similar to Master/Worker	High	Yes	Yes	Yes	Low/Sub second	Open Source
APACHE SAMZA	Kafka Architecture	High	Yes	Yes	Yes	Low/Sub second	Open Source
APACHE KUDU	Master/Worker	High	Yes	Yes/Query	Yes	Low	Open Source
APACHE NIFI	Not Master/Worker	High	Yes	Yes	No	Low	Open Source
APACHE KAFKA	Publish/Subscribe	high	Yes	Yes	Yes	Low	Open Source
AMAZON KINESIS	Publish/Subscribe	Depends on no. of shards	yes	Redshift, S3 etc	Supports highly redundant network	Low/Sub second	Commercial Service/Some open source libraries

6. Conclusion

This research focused on real-time stream processing of big data with emphasis on the commonly used frameworks for real time stream processing. Real time streaming is becoming more eventful, which makes its relevance in our daily life very important. This expanding era of automatic data analysis is proving to be beneficial in all works of life such as business, health care, banking and finance, manufacturing, climate and weather, fraud and forensic studies etc. It is a reflection of the importance of data in real-time, uniquely showing the worth of distributed real time streaming technologies. This literature has also introduced the two important architectures concerned with real time processing high-lighting both their advantages and draw backs. This literature ends with Table 1comparing and contrasting the features of eight distributed real-time systems.

References

[1] Bansal, Ankita & Jain, Roopal & Modi, Kanika. (2019). Big Data Streaming with Spark
[2] Nicoleta Tantalaki, Stavros Souravlas & Manos Roumeliotis (2019): A review on big data real-time stream processing and its scheduling techniques, International Journal of Parallel, Emergent and Distributed Systems
[3] Apache Spark™—Unified Analytics Engine for Big Data. [Online]. Available: https://spark.apache.org/ [Accessed Oct. 2020]
[4] Namiot, Dmitry. (2015).On Big Data Stream Processing. International Journal of Open Information Technologies. 3. 48-5
[5] D.Peng and F.Dabek. Large scale incremental processing using distributed transactions and notifcations. In Proceedings of the 9th USENIX Symposium on Operating Systems Design and Implementation,2010
[6] Y. Yorozu, M. Hirano, K. Oka, and Y. Tagawa, "Electron spectroscopy studies on magneto-optical media and plastic substrate interface," IEEE Transl. J. Magn. Japan, vol. 2, pp. 740–741, August 1987 [Digests 9th Annual Conf. Magnetics Japan, p. 301, 1982].
[7] Wingerath, Wolfram & Gessert, Felix & Friedrich, Steffen & Ritter, Norbert. (2016).Real-time stream processing for Big Data. - Information Technology.
[8] M. Zaharia, T. Das, H. Li, et al. Discretized streams: Fault tolerant streaming computation at scale. In Proceedings of the Twenty-Fourth ACM Symposium on Operating Systems Principles, SOSP '13, pages 423–438, New York, NY, USA, 2013.
[9] Apache Storm. [Online]. Available: https://storm.apache.org/ [Accessed October 2020]
[10] Nasiri, H., Nasehi, S. & Goudarzi, M. Evaluation of distributed stream processing frameworks for IoT applications in Smart Cities. J Big Data 6, 52 (2019).
[11] Samza-Architecture [Online]. Available: http://samza.apache.org/learn/documentation/latest/introduction/architecture [Accessed Oct. 2020]
[12] Lambda architecture [Online].Available: http://lambda-architecture.net/ [Retrieved: Sep. 2020]
[13] Apache Flink [Online]. Available: https://flink.apache.org/flinkarchitecture. [Retrieved: Sep.2020]

[14] Apache Flume [Online]. Available: https://flume.apache.org/ [Retrieved: Sep.2020]
[15] J.Kreps“, Questioning the Lambda Architecture” [Online]. Available: https://www.oreilly.com/radar/questioning-the-lambda-architecture/ [Acessed Sep.2020]
[16] Zschörnig, Theo & Wehlitz, Robert & Franczyk, Bogdan. (2017). A Personal Analytics Platform for the Internet of Things - Implementing Kappa Architecture with Microservice-based Stream Processing.
[17] Spark Streaming https://spark.apache.org/docs/latest/streaming-programming-guide.html [Retrieved Oct.2020]
[18] Logs and Real Time Stream Processing [Online]. Available: https://www.oreilly.com/content/i-heart-logs-realtime-stream-processing/ [Retrieved: Oct. 2020]
[19] ci.apache.org [Online]. Available: [Retrieved: Oct. 2020]
[20] Top ten big data framework in 2020 | Jelvix https://jelvix.com/blog/top-5-big-data-frameworks [Retrieved: Oct. 2020]

Author Profile

Denis P. Bell received the B.Eng. in Computer Science from Zhejiang University of Science and Technology in 2018. He is currently pursuing a master’s degree in Advance manufacturing and Informatization in the same university. His interests span database technologies, big data, web design & development and computer networking.

EliasuTambominyi received the B.Eng. in Computer Science from Zhejiang University of Science and Technology in 2018. He is currently pursuing a master’s degree in Intelligent Manufacturing in the same university. His interests revolve around big data, machine learning, neural networks and information security. With a background in software and mobile developing, he is a keen web and mobile developer with an apt for detecting and solving problems associated with web and mobile applications
.