Advances and Applications in Mathematical Sciences
Volume 19, Issue 6, April 2020, Pages 471-479
© 2020 Mili Publications

A NOVEL STREAM PROCESSING FRAMEWORK FOR FASTER DATA PROCESSING

KAMAL KUMAR RANGA and CHANDER KUMAR NAGPAL

Department of Computer Engineering
YMCA University of Science and Technology
Faridabad, Haryana, India
E-Mail: kamal.ranga@gmail.com
nagpalckumar@rediffmail.com

Abstract

Big data which started with merely fulfilling increasing need of storage has kicked the development in computing world by making scientists more and more hungry to get and give more. Even though, we have much faster computer systems today, still the speed at which data is growing is really lightening, thus to match up, processing speed must also be such that the incoming data would be processing as it comes into system [11] [13].

To cater this we have various stream processing frameworks, like apache Spark, Storm and many more development by companies for their own use, still we need a more faster system to process data for which we have proposed a framework, which would fusion multiple data and clean useless data using fuzzy rule based system.

In this paper, the authors have researched about why there is preference of stream over batch processing, then they have studies various models available to process streams and compared them on various parameters. Later, outlined need of a good processing framework and proposed a framework for stream processing.

1. Introduction

Stream processing is a big data technology that enables processing continuous stream of data, here processing includes querying, detecting and identifying patterns within minimum possible time (ms to min).For e.g.: Receiving an alert as temperature reaches certain point and querying streams received from the sensor [2].

2010 Mathematics Subject Classification: 62-07, 97R50.
Keywords: Batch, Stream, Framework, Processing, Latency, Reliability, Cleaning.
Received 31 October 2019; Accepted 4 December 2019

Why is stream processing needed?

As Big data mainly focuses on gathering maximum value insights by processing data which may not be created equal depending upon the quality of data, i.e. some data is of more value than other and also the value ceases rapidly with time [1]. Thus processing must be as fast as possible to retain maximum value, which can done by stream processing by enabling faster insights, often within ms to s from trigger.

For data that comes as never-ending stream, if we batch process, we need to store then at some point stop data collection and process data. In this process we definitely miss sufficient amount of data and further, processing is altogether very slow [3][4]. Further, for next batch aggregation across multiple batches would be our main concern.

On the other hand, streaming handles such data gracefully by detecting patterns, inspecting results, focusing on multiple levels and also can cater multiple streams simultaneously and gives result within very less time without any loss of information, e.g., Health and traffic sensors, activity and transaction logs and IoT data (almost all).

1. Batch processing collects data first and then processes it, while stream processing process data as it comes, which makes stream processing to work with least hardware requirements [6].

2. Due to systematic load shedding stream processing empowers approximate query processing which is not the case with batch processing.

3. Continuous data is generally huge and we are unable to store it. Stream processing makes us capable of handling this data wisely and retains only useful bits [10].

4. With huge streaming data available (e.g. website visits, customer transactions and activities) and it will definitely grow exponentially with spread of IoT and computing technologies, stream processing is much more natural model to think [7][8].

However, Stream Processing is not suitable for all sort of data. We can make a rule that if processing would be possible in single pass over data or is locally available then it would best fit for stream processing.

2. Stream Processing Systems

Stream processing systems operates over data as it gets into system, whose processing is a lot different than batch processing. Where we defines operations to be performed onto each individual data item rather than complete data set. In stream processing the data sets are unbounded i.e. data is coming from multiple sources and in different formats [9] [11] [14].

Stream processing handles almost unlimited data, but the processing is done either of the two ways:

(a) Micro-batching: Here the system processes very few data items like from tens to hundreds of data items at a time.

(b) True Stream: Here system only processes exactly one data item at a time.

Also, very minimum state is being maintained in between data items in both the cases. There are few implications of data sets:

- Amount of data entered into system till an instant is called Total dataset.

- At an instant a single data item is processed at a time, this is called working dataset.

- Stream processing provides event-based processing and ends only when stopped explicitly.

- Result of processing are available instantly and are updated continuously as new data arrives.

Streaming model offers nearly real-time processing requirements. As sit involves reacting to the changes in the system instantly the applications such as error logging for server or application, analysis of data, heat sensors or satellite propagation and other time-based metrics are a natural fit [8] [14].

There exists two most commonly and widely used framework for stream processing

(a) Apache Storm.

(b) Apache Samza.

3. Comparison among various Frameworks

Here we are comparing apache storm and Apache Spark on various desired services.

Table 1. Comparison between Apache Spark and Apache Storm.

Feature	Apache Storm	Apache Spark
Stream Processing	Micro-batch processing	Batch processing
Stream Sources	Spout	HDFS
Language Support	Java, Scala, and Clojure (Scala supports multiple languages)	Java, Scala (Scala supports fewer languages)
Latency	Low latency	High
Messaging	ZeroMQ, Netty	Akka, Netty
Persistence	MapState	Per RDD
State Management	Apache Storm supports state management with slight increase in latency.	Apache Spark also supports state management
Provisioning	Apache Ambari	Basic, using Ganglia
Reliability	At least Once and Exactly Once.	Only Exactly once mode.
Throughput	10k records per node per second	100k records/node /second
Fault Tolerance	Fail-fast and stateless.	Replication and Check pointing.

4. Literature Review

[APR, 2018] The author(s) here raises concern over the raw data being highly unstructured and heterogeneous, thus valuable information must be kept locally. For this they have proposed a fusion of 3 different data models but it have very limited querying capabilities [2].

[MAR, 2019] Author(s) here says that due to uneven task execution in stream processing, latency sensitive applications suffers. They presented a pre scheduling framework called lever that is an extension to apache spark, which evaluates the capacity of each node and preschedules jobs [9].

[JUNE, 2018] Here authors have presented a Hierarchically Distributed Data Matrix (HDM) framework to run on distributed system. It showed improvement but requires a lot of work to be done on heterogeneity, fault tolerance and memory [4].

[DEC, 2016] Here author(s) are concerned about non-availability of time critical big data framework, and thus presented a framework which is suitable for only small amount of data and it lacks completeness too [8].

[March, 2019] Author(s) here says that with increase in amount of data there must be increase in speed of real-time processing of data and prioritize the urgent need of such framework with low latency [6].

[Jan, 2019] Author(s) here quoted that highly dynamic and intense data streams are faced by stream processing systems, with parallelization and elasticity such streams are capable of providing high quality of service [7].

5. Proposed Architecture

As can be seen from the block diagram below, we are focussed to work on three main changes to the traditional architecture of stream processing.

Data Fusion Layer: This layer is used as a feeding system that will accept data from multiple heterogeneous sources and then is fusioned such that data is fed irrespective of its type. Further, in fusion of multiple data from multiple sources and of heterogeneous type, it might be thought that to maintain uniformity in incoming data we may introduce a system that converts incoming data into bit stream which simplify the system and helps integrating heterogeneous data as a single homogenous data [15].

But practically introduction of such system will lead to extreme over to the system as this introduces several overheads:

1. Converting all incoming data into bit stream in itself a very tedious and bulky task.

2. This will severely affect the throughput of the system and response time to great extent.

This will slow down system and rather than gaining any benefit it would proved to be a bulky and slow system.

Thus, to effective increase the speed of system we must be focussed to keep the system as light and simple as possible.

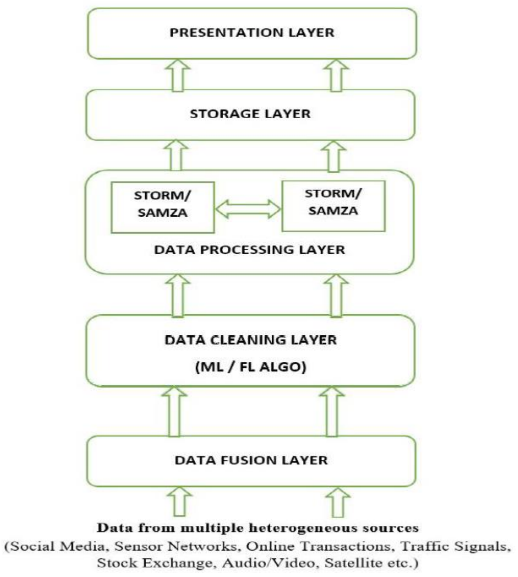
PRESENTATION LAYER
STORAGE LAYER
STORM/ SAMZA
STORM/ SAMZA
DATA PROCESSING LAYER
DATA CLEANING LAYER
(ML / FL ALGO)
DATA FUSION LAYER
Data from multiple heterogeneous sources
(Social Media, Sensor Networks, Online Transactions, Traffic Signals, Stock Exchange, Audio/Video, Satellite etc.)

Figure 1. Proposed Stream Processing Architecture showing layers required to process data faster.

Data Cleaning Layer: As most data entering into system might not be of use, thus it becomes very crucial to design a data cleaning algorithm in such a way that only the data that would be of value in any later stage of processing enter the system and all other data which is practically non usable should be truncated at this stage. For this we should train our system only to accept data which is of importance to further stages. Thus, we need to design a good machine learning or fuzzy rule based system for same [11] [15].

The main benefit of doing this would be:

(a) The storage layer would now get only required data not all incoming traffic.

(b) This would decrease latency as there is no need to store all garbage (unusable data) and only cleaned data is passes to next layer.

Real-Time Processing Layer: In this layer actual data processing is carried out by either Storm/ Samza. (Storm offers extremely low latency while samza offers feature to store intermediate results which might provide better results. Thus, the decision of using either of the framework is left for implementation stage).

As with our new architecture we are focused to improve latency further, this can be achieved by applying machine learning or fuzzy logic algorithms in a way that the processing is done with improved speed [17] [18].

Here we can use spout (producer of stream) and bolts (small functions) as described in storm stream processing in an efficient manner.

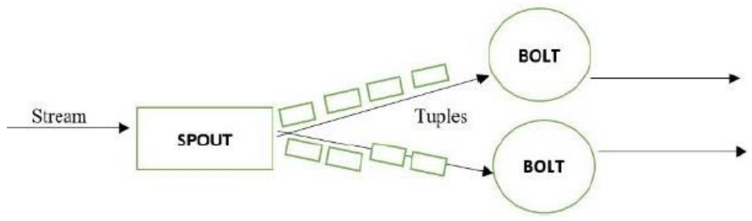

Figure 2. Spout and Bolt in action (Simple Way).

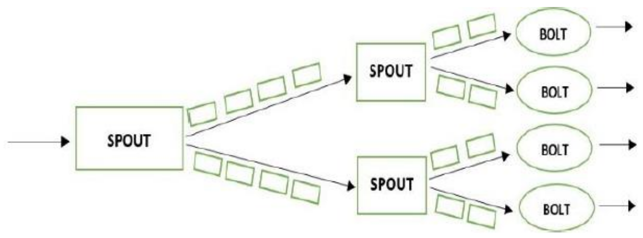

Figure 3. Spout and Bolt in action (Other Way).

Storage Layer: Storage has always been a challenge as the size of data we are dealing with, becomes large. Here this challenge is catered by storing only the result of processing which would automatically rule out the data which is of no use for our framework in the previous step i.e. processing.

Although, finding a storage solution is still very much important. This layer focuses on to store data efficiently [4] [6].

Presentation Layer: This layer is to present the result into suitable format and to analyse if required further. The result can be used by visualization applications, human beings, business processes, or services [3] [15]. (Presentation and analysis at this layer is beyond the scope of our work).

6. Conclusion

Through the above papers as we have seen that, although much work has been done to improve currently available frameworks still the exists a scope for further refinement in the framework so as to make it fit for scale, clean and lower latency as amount of data, speed of data and sources from which it come will increase drastically in near future.

The proposed framework will be able to collect data from multiple sources, integrate them and clean maximum data which is irrelevant to our system. Further, it will also capable of having faster processing and lower latency to offer so as to match and excel among existing frameworks available right now.

We also have planned to improve the processing capability of the framework by applying machine learning/ fuzzy logic that will help in better and faster processing of data and would further improve the latency and efficiency of the proposed framework.

References

[1] L. Affetti, R. Tommasini, A. Margara, et al., Defining the execution semantics of stream processing engines, J Big Data 4 (2017), 12.

[2] M. D. Assuncao, A. D. Veith and R. Buyya, Distributed data stream processing and edge computing: A survey on resource elasticity and future directions, Journal of Network and Computer Applications 103 (2018), 1-17.

[3] D. Cheng, Y. Chen, X. Zhou et al. Adaptive Scheduling of Parallel Jobs in Spark Streaming, IEEE INFOCOM 2017- IEEE Conference on Computer Communications; 2017; Atlanta, GA, Pg 1-9.

[4] Dongyao Wuy, Sherif Sakrz, Liming Zhuy and Qinghua Lux, HDM: A Composable Framework for Big Data Processing, IEEE Trans. on Big Data 4(2) (06-2018), 150-163.

[5] G. M. Evgenyevich, B. A. Valerievich and B. M. Alekseevna, Using apache spark to collect analytic from the streaming data processing application logs, 7th Mediterranean Conference on Embedded Computing (MECO), Budva, 2018, pp. 1-4.

[6] Hai Jin, Fei Chen, Song Wu, Yin Yao, Zhiyi Liu, Lin Gu and Yongluan Zhou, Towards Low-Latency Batched Stream Processing by Pre-Scheduling, IEEE Transactions on Parallel and Distributed Systems, Page(s) (03-2019), 710-722.

[7] Henriette Roger and Ruben Mayer, A Comprehensive Survey on Parallelization and Elasticity in Stream Processing, ACM Computing Surveys, ACM Digital Library Volume 49, Issue 1 (01-2019).

[8] Martin Hirzel, Robert Soulé, Scott Schneider, BuğraGedik and Robert Grimm, A catalog of stream processing optimizations, ACM Computing Surveys, ACM Digital Library Volume 46, Issue 4 (12-2016).

[9] Nicoleta Tantalaki, Stavros Souravlas and Manos Roumeliotis, A review on big data real-time stream processing and its scheduling techniques, Taylor and Francis-International Journal of Parallel, Emergent and Distributed Systems. (03-2019).

[10] P. Basanta-Val, N. C. Audsley, A. J.Wellings, I. Gray and N. Fernandez, Architecting Time-Critical Big-Data Systems, IEEE Trans. on Big Data, Vol 2, Issue 4 (12-2016).

[11] Salman Salloum, Joshua Zhexue Huang, Yulin He and Xiaojun Chen, An Asymptotic Ensemble Learning Framework for Big Data Analysis, IEEE Access, Volume 7. (12-2018),

[12] P. Smirnov, M. Melnik and D. Nasonov, Performance-aware scheduling of streaming applications using genetic algorithm, Procedia Computer Science, Volume 108 (2017), 2240-2249.

[13] Saumya Ounacer, Md. Amin Talhaoui, Soufiane Ardchir, Abderrahmane and Md. Azouazi, A new architecture for real time data stream processing, International Journal of Advanced Computer Science and Applications, Volume 8, Issue 11. (11-2017),

[14] Soumaya Ounacer, Md. Amine Talhaoui, Soufiane Ardchir, Abderrahmane Daif and Md. Azouazi, Real-time Data Stream Processing Challenges and Perspectives, IJCSI International Journal of Computer Science Issues, Volume 14, Issue 5, (09-2017).

[15] Sohail Jabbar, Kaleem R. Malik, Mudassar Ahmad, Omer Aldabbas, Muhammad Asif, Shehzad Khalid, Ki J. Han and Syed H. Ahmad, A Methodology of Real-Time Data Fusion for Localized Big Data Analytics, IEEE Access, Volume 6 (04-2018),24510-24520.

[16] Stefanos Antaris and Dimitrios Rafailidis, In-Memory Stream Indexing of Massive and Fast Incoming Multimedia Content, IEEE Transactions on Big Data 4(1) (03-2018), 40-54.

[17] The Apache Software Foundation, Apache Spark [Internet], 2018, Available from http://spark.apache.org/.

[18] Xunyun Liu and Raj Kumar Buyya, The University Of Melbourne, Australia, Resource Management and Scheduling in Distributed Stream Processing Systems: A Taxonomy, Review and Future Directions, ACM Computing Surveys, ACM Digital Library 48(1) (03-2018).