On Big Data Stream Processing

Dmitry Namiot

***Abstract*—In this paper, we would like to discuss data stream processing in the big data area. Our goal is to provide a quick introduction and survey of the technical solutions for big data streams processing. In this survey, we target Machine to Machine communications, sensors fusion in Internet of Things as well as time series data processing. We discuss the basic elements behind data streams processing, the existing technical solutions for their implementations as well some prospect system architectures.**

***Keywords*—streams, Spark, time series, sensors.**

I. Introduction

This paper continues our series of papers devoted to Machine to Machine (M2M) [1] and Internet of Things processing (IoT) [2].

A sensor network (as a typical form in IoT applications) consists of small computational devices that are able to communicate over wireless connection channels [3]. Each of these computational devices is equipped with sensing, processing and communication facilities. Actually, the sensor networks will form a new world wide web that can read the physical world in real time. They are generating data streams that need to be processed in real time for a wide range of applications in the various areas.

Various data streams could have own features. For example, the data stream from the financial market describes the whole data. In the same time, the data stream for sensors depends on sampling (e.g., we can get new data every 5 minutes) and so, presents a sample of the entire population. Sometimes, data streams could be noisy. Spatial and temporal attributes could play an important role in data streams processing. In many cases (e.g., in sensor networks) we have to pay attention the limited resources (e.g., space and energy) for data streams processing. Real-time data streams processing could add own complexity too.

We can highlight two major tasks for data stream processing. At the first hand, it is processing queries for data streams [4]. Event processing [5] for data streams falls into this category. And the second big category is data mining. We can mention here clustering [6,7], classification and prediction [8,9], time series [10] and change detection [11], frequent pattern [12] and outlier detection [13]. Data mining for stream processing is, probably, the most actively growing direction.

Of course, the above-mentioned tasks require technological (IT) support. Software tools for big data stream processing [14] are a subject of this paper.

D.Namiot is with the Lomonosov Moscow State University (e-mail: dnamiot@gmail.com).

The rest of this paper is organized as follows. In Section II, we discuss the basics of stream processing. In Section III, we present a survey of modern approaches for big data stream processing. In Section IV, we discuss lambda architecture.

II. Stream processing

In a formal way, a data stream is any ordered pair (S,*T*) where:

- *S* is a sequence of tuples and
- *T* is a sequence of positive real time intervals.

So, it defines a data stream as a sequence of data objects. The sequence in a data stream is potentially unbounded. It means that data streams may be continuously generated at any rate. Indeed, many sources may produce data continuously. Examples include a machine to machine communications (M2M), Internet of Things (IoT) objects, sensor networks, tags (beacons), etc. It is very important also, that an ordered sequence of instances in data streams can be read only once or a small number of times. This reading process very often should use the limited computing and storage capabilities (it is especially true for the modern big data streams). In the data stream, each data object can be described by a multidimensional attribute vector within a continuous, categorical, or mixed attribute space [15]. There are some typical characteristics of data streams:

- Continuous arrival of data objects
- Disordered arrival of data objects
- Potentially unbounded size of a stream

Data streams can be generated in various scenarios, including a network of sensor nodes, a stock market or a network monitoring system and so on.

There are several important queries to be considered [17]:

- Aggregate Queries. Aggregate Queries is an important class of queries in sensor systems, including MIN, COUNT and AVG operators.

- Join Queries. An example of join queries is “Return the objects that were detected in both regions R1 and R2”. To evaluate the query, stream readings from the sensors in regions R1 and R2 should be joined first before we can determine whether an object was detected in the two designated regions. Join queries are useful in many applications, such as monitoring, where multiple devices (e.g., sensors) provide measurements data.

- Continuous Queries. To monitor designated changes in data are typically required to answer queries in a continuous manner. For example, when the query constraints are satisfied, the designated action could be triggered.

Data stream mining can extract useful rules/information from data streams. The following lists some typical tasks for stream mining:

- Clustering. Clustering is the task of grouping a set of objects in such a way that objects in the same group (called a cluster) are more similar to each other than to those in other groups (clusters). Clustering techniques for data streams typically continuously cluster objects on memory constrained devices with some time limitations.

- Classification. Classification uses prior knowledge to guide the partitioning process to construct a set of classifiers to represent the possible distribution of patterns. Basically, compared with clustering, classification is a supervised learning process whereas clustering is an unsupervised learning process. More formally, a typical classification algorithm can be defined as follows [18]: given a predefined classifier and two sets of data, labeled data and unlabeled data, the labeled data is used to train the classifier and the unlabeled data can then be classified by the trained classifier.

- Frequent Items Mining. Frequent items mining is to find sets of items or values that co-occur frequently, or in other words, to find co-occurrence relationships in a data set where a set of items appears together in some specified context.

- Outlier and Anomaly Detection. In outlier and anomaly detection, the main task is to find data points that are most different from the remaining points in a given data set. Most existing outlier detection algorithms are based on the distance between every pair of points. The points that are most distant from all other points will be marked as outliers.

A stream processing solution has to solve different challenges [19]:

- Processing massive amounts of streaming events (filter, aggregate, rule, automate, predict, act, monitor, alert)
- Real-time responsiveness to changing market conditions
- Performance and scalability as data volumes increase in size and complexity
- Rapid integration with existing infrastructure and data sources: Input (e.g. market data, user inputs, files, history data from a DWH) and output (e.g. trades, email alerts, dashboards, automated reactions)
- Fast time-to-market for application development and deployment due to quickly changing landscape and requirements
- Developer productivity throughout all stages of the application development lifecycle by offering good tool support and agile development
- Analytics: Live data discovery and monitoring, continuous query processing, automated alerts and reactions
- Community (component / connector exchange, education / discussion, training / certification)
- End-user ad-hoc continuous query access
- Alerting
- Push-based visualization

III. BIG DATA STREAMS

In this section, we discuss some technological solutions for data streams processing.

Apache Storm is a distributed real-time computation system for processing large volumes of high-velocity data [20]. Is a distributed real-time computation system for processing fast, large streams of data. Storm is an architecture based on master-workers paradigm. So a Storm cluster mainly consists of a master and worker nodes, with coordination done by Zookeeper.

Spark Streaming [21] is an extension of the core Spark API [22] 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 1).

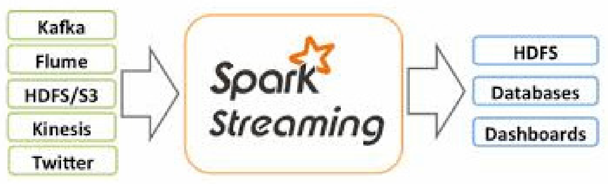

Fig. 1 Spark Streaming

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 streams (Figure 2).

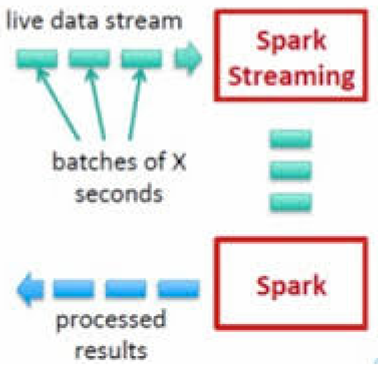

Fig. 2 Spark processing

Apache Samza [23] is a distributed stream processing framework. It uses Apache Kafka for messaging, and Apache Hadoop YARN to provide fault tolerance, processor isolation, security, and resource management (Figure 3).

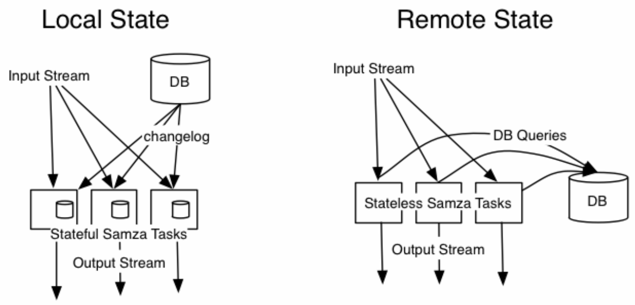

Fig. 3. Apache Samza

Apache Flume [24] is a distributed, reliable, and available service for efficiently collecting, aggregating, and moving large amounts of log data. It has a simple and flexible architecture based on streaming data flows. It is robust and fault tolerant with tunable reliability mechanisms and many failovers and recovery mechanisms. It uses a simple extensible data model that supports online analytic applications (Figure 4).

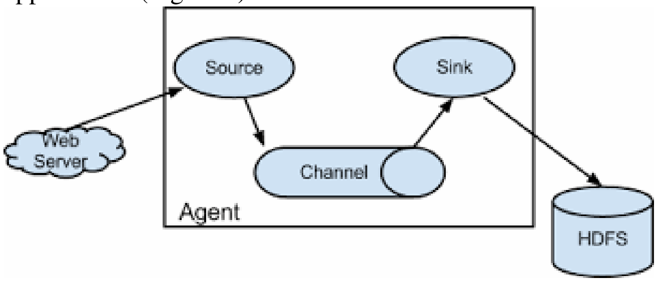

Fig. 4. Apache Flume

Apache Kafka itself is often used as a kernel for data stream architecture. Originally, Apache Kafka is publish-subscribe messaging rethought as a distributed commit log [25]. Apache Kafka is a distributed system designed for streams. It is built to be fault-tolerant, high-throughput, horizontally scalable, and allows geographically distributing data streams and processing. Figure 5 illustrates stream-centric architecture in Linkedin [26]

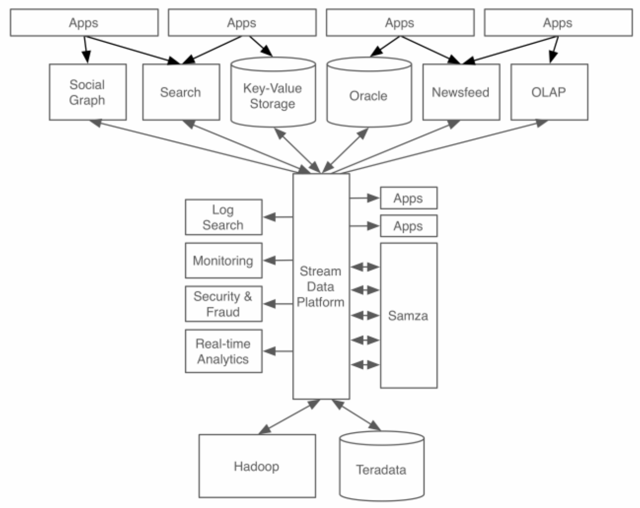

Fig. 5 Stream-centric architecture on Apache Kafka [26]

Amazon Kinesis [27] is a fully managed, cloud-based service for real-time data processing over large, distributed data streams. Amazon Kinesis can continuously capture and store terabytes of data per hour from hundreds of thousands of sources such as website clickstreams, financial transactions, social media feeds, IT logs, and location-tracking events.

IBM InfoSphere Streams [28] is an advanced analytic platform that allows user-developed applications to quickly ingest, analyze and correlate information as it arrives from thousands of real-time sources. The solution can handle very high data throughput rates, up to millions of events or messages per second (Figure 6).

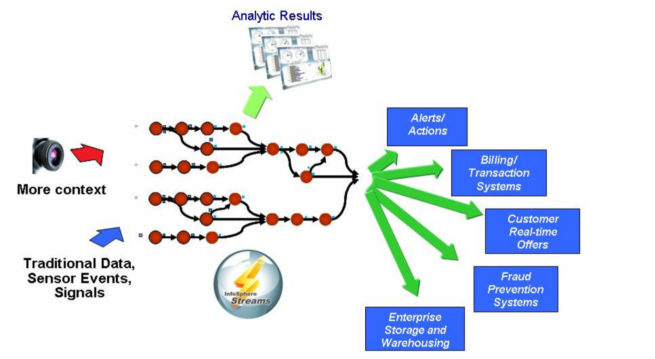
Analytic Results
More context
Traditional Data, Sensor Events, Signals
InfoSphere Streams
Alerts/ Actions
Billing/ Transaction Systems
Customer Real-time Offers
Fraud Prevention Systems
Enterprise Storage and Warehousing

Fig. 6 IBM InfoSphere Streams

As per IBM's benchmark, this solution outperforms Apache Storm [29].

The TIBCO StreamBase® Complex Event Processing (CEP) platform is a high-performance system for rapidly building applications that analyze and act on real-time streaming data [30].

IV. Lambda architecture

The Lambda Architecture is an approach to building stream processing applications on top of MapReduce and Storm or similar systems (Figure 7).

The way this works is that an immutable sequence of records is captured and fed into a batch system and a stream processing system in parallel. Developers implement business transformation logic twice, once in the batch system and once in the stream processing system. It is possible to combine the results from both systems at query time to produce a complete answer [31].

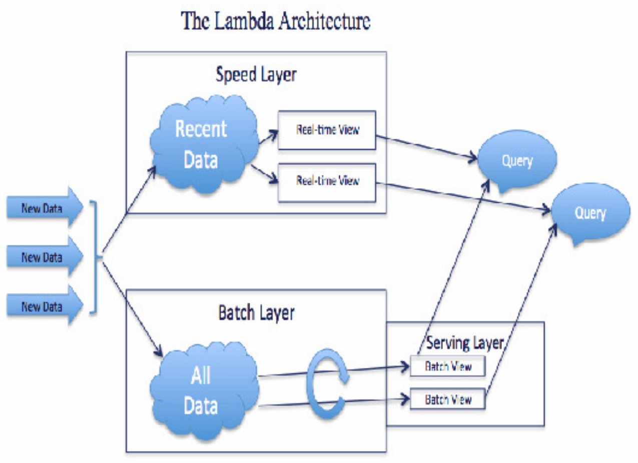

Fig. 7. Lambda architecture [32]

The Lambda Architecture is aimed at applications built around complex asynchronous transformations that need to run with low latency. The problem with batch processing is the time it takes. In the meantime, data has been arriving and subsequent processes or services continue to work with old information. The dedicated real time layer solves this by taking its copy of the data, processing it quickly and stores it in a fast store. This store is more complex since it has to be constantly updated.

But there are disadvantages too. One of the obvious remarks is the need for duplicating business rules. Practically, the developers need to write the same code twice – for real-time and batch layers. One proposed approach to fixing this is to have a language or framework that abstracts over both the real-time and batch framework [33]. The proposed solution is Summingbird framework [34]. It is a library that lets you write MapReduce programs that look like native Scala or Java collection transformations and execute them on a number of well-known distributed MapReduce platforms. In other words, the same code could be executed on both layers in lambda architecture.

V. Conclusion

In this short paper, we provide an introduction for stream processing in a big data area. We are planning to provide a more deep analysis for the above-mentioned systems in the upcoming papers. By our opinion, real time data processing is a key area for IoT and M2M applications.

References

[1] Namiot, D., & Sneps-Sneppe, M. (2014). On M2M Software Platforms. International Journal of Open Information Technologies, 2(8), 29-33.

[2] Namiot, D., & Sneps-Sneppe, M. (2014). On IoT Programming. International Journal of Open Information Technologies, 2(10), 25-28.

[3] Gama J., and Gaber MM (Eds), Learning from Data Streams: Processing Techniques in Sensor Networks, Springer Verlag, 2007

[4] Aggarwal, C. C. (2007). Data streams: models and algorithms (Vol. 31). Springer Science & Business Media.

[5] Cugola, G., & Margara, A. (2012). Processing flows of information: From data stream to complex event processing. ACM Computing Surveys (CSUR), 44(3), 15.

[6] Liu, Y. B., Cai, J. R., Yin, J., & Fu, A. W. C. (2008). Clustering text data streams. Journal of computer science and technology, 23(1), 112-128.

[7] Gama, J. (2010). Clustering from Data Streams. In Encyclopedia of Machine Learning (pp. 180-183). Springer US.

[8] Aggarwal, C. C., Han, J., Wang, J., & Yu, P. S. (2004, August). On demand classification of data streams. In Proceedings of the tenth ACM SIGKDD international conference on Knowledge discovery and data mining (pp. 503-508). ACM.

[9] Yang, Y., Wu, X., & Zhu, X. (2005, August). Combining proactive and reactive predictions for data streams. In Proceedings of the eleventh ACM SIGKDD international conference on Knowledge discovery in data mining (pp. 710-715). ACM.

[10] Hulten, G., Spencer, L., & Domingos, P. (2001, August). Mining time-changing data streams. In Proceedings of the seventh ACM SIGKDD international conference on Knowledge discovery and data mining (pp. 97-106). ACM.

[11] Kifer, D., Ben-David, S., & Gehrke, J. (2004, August). Detecting change in data streams. In Proceedings of the Thirtieth international conference on Very large data bases-Volume 30 (pp. 180-191). VLDB Endowment.

[12] Giannella, C., Han, J., Pei, J., Yan, X., & Yu, P. S. (2003). Mining frequent patterns in data streams at multiple time granularities. Next generation data mining, 212, 191-212.

[13] Subramaniam, S., Palpanas, T., Papadopoulos, D., Kalogeraki, V., & Gunopulos, D. (2006, September). Online outlier detection in sensor data using non-parametric models. In Proceedings of the 32nd international conference on Very large data bases (pp. 187-198). VLDB Endowment.

[14] Stonebraker, M., Çetintemel, U., & Zdonik, S. (2005). The 8 requirements of real-time stream processing. ACM SIGMOD Record, 34(4), 42-47.

[15] Silva, J. A., Faria, E. R., Barros, R. C., Hruschka, E. R., de Carvalho, A. C., & Gama, J. (2013). Data stream clustering: A survey. ACM Computing Surveys (CSUR), 46(1), 13.

[16] Qin, Y., Sheng, Q. Z., Falkner, N. J., Dustdar, S., & Wang, H. (2014). When Things Matter: A Data-Centric View of the Internet of Things. arXiv preprint arXiv:1407.2704.

[17] Subramaniam, S., & Gunopulos, D. (2007). A survey of stream processing problems and techniques in sensor networks. In Data Streams (pp. 333-352). Springer US.

[18] Wang, F. and Liu, J. 2011. Networked Wireless Sensor Data Collection: Issues, Challenges, and Approaches. IEEE Communications Surveys and Tutorials 13, 4, 673–687

[19] Real-Time Stream Processing as Game Changer in a Big Data World with Hadoop and Data Warehouse http://www.infoq.com/articles/stream-processing-hadoop Retrieved: Jul, 2015

[20] Jain, A., & Nalya, A. (2014). Learning Storm. Packt Publ..

[21] Spark Streaming http://spark.apache.org/docs/latest/streaming-programming-guide.html Retrieved: Jul, 2015

[22] Shoro, A. G., & Soomro, T. R. (2015). Big Data Analysis: Apache Spark Perspective. Global Journal of Computer Science and Technology, 15(1).

[23] Apache Samza http://samza.apache.org/ Retrieved: Jul, 2015

[24] Apache Flume http://flume.apache.org/ Retrieved: Jul, 2015

[25] Kafka, A. (2014). A high-throughput, distributed messaging system. URL: kafka. apache. org as of, 5(1).

[26] Putting Apache Kafka To Use: A Practical Guide to Building a Stream Data Platform (Part 1) http://www.confluent.io/blog/stream-data-platform-1/ Retrieved: Jul, 2015

[27] Amazon Kinesis http://aws.amazon.com/kinesis/ Retrieved: Jul, 2015

[28] Ballard, C., Brandt, O., Devaraju, B., Farrell, D., Foster, K., Howard, C., ... & Uleman, R. (2014). Ibm Infosphere Streams: Accelerating Deployments with Analytic Accelerators. IBM Redbooks.

[29] Of Streams and Storms https://developer.ibm.com/streamsdev/wp-content/uploads/sites/15/2014/04/Streams-and-Storm-April-2014-Final.pdf Retrieved: Jul, 2015

[30] The TIBCO StreamBase Complex Event Processing http://www.tibco.com/products/event-processing/complex-event-processing/streambase-complex-event-processing Retrieved: Jul, 2015

[31] Lambda architecture http://lambda-architecture.net/ Retrieved: Jul, 2015

[32] Simplifying the (complex) Lambda architecture http://voltdb.com/blog/simplifying-complex-lambda-architecture Retrived: Jul, 2015.

[33] Questioning the Lambda Architecture http://radar.oreilly.com/2014/07/questioning-the-lambda-architecture.html Retrieved: Jul, 2015

[34] Boykin, O., Ritchie, S., O'Connell, I., & Lin, J. (2014). Summingbird: A framework for integrating batch and online mapreduce computations. Proceedings of the VLDB Endowment, 7(13), 1441-1451.