A REVIEW OF STREAM PROCESSING PLATFORMS AND TECHNIQUES

[1]Sindhu B Dinesh, [2]Dr. Hemavathy R
[1]Student, [2]Associate Professor
Department of Computer Science, RVCE, Bangalore, India

Abstract: In the current world, stream processing is considered as one of the big data technologies. Distributed stream processing platforms are real-time monitoring systems that help evaluate, analyze and infer insights from huge unbounded data streams in an efficient way. Since these are characterized by high throughput and low latency, they suffice the requirements of Big Data or Internet of Things applications. However, these platforms offer services that are raw and hence make it difficult to integrate a real-world application with them. Moreover, given the options, choosing the right platform requires some effort. This paper analyses and describes the architectures, features such as fault tolerance, recovery, efficiency and throughput of some stream processing platforms. The study shows that native stream processing systems perform better than micro-batching. Owing to the high-speed evolving nature of this data, several challenges are faced in mining it. This paper also throws a light on such challenges and presents some techniques in mining such data. The performance is compared and analyzed based on the certain evaluation metrics.

***Index Terms* - Stream processing, fault tolerance, stream processing platforms and techniques**

I. INTRODUCTION

The huge amount of big data from sources such as senor monitoring, network traffic analysis, security threat detection, e-commerce, health industry and cloud management can help provide insights in various of public life, such as security, medicine, environment, and smarter use of cities, but is rarely exploited due to aspects of privacy, ethics and availability of appropriate technology, among the others. The streaming data is continuous and unbounded and needs to be processed in real-time to be effective and useful applications like Internet of Things. The Big data platforms provide services to handle the same but are raw and thus, ways to exploit them for stream processing is done in this paper. The goal of such analysis is the discovery of new and hidden information from the output of data sources as they occur, thus with very limited latency. Such technologies have to meet strict requirements: rapid processing, high accuracy, and minimal human intervention. In this context, Big Data techniques can help this automated analysis, e.g., to enable corrective actions or to signal a security alarm for citizens.

Owing to the high-speed unbounded nature of stream processing applications, real-time processing cannot be done using conventional centralized systems. Hence, the requirements for stream processing are different from those of batch processing. The approximate sensors by 2025 is around 80 billion [2]. This quantity of data has to processed by big data platforms and not by centralized systems. However, Hadoop processing is generally on synchronous data persisted in a passive data store. Processing and monitoring in real-time generally analyses various stream sources and produce large number of alerts to the systems in real-time. This kind of processing also makes use of historic data, in learning and model building. These requirements translate into some general-purpose platforms that offer high availability, throughput, scalability and low latency. This studies the systems like the Apache Samza, Apache S4, Apache Storm [3] and the Apache Flink [4], and one micro-batch system, the Apache Spark Streaming [6]. The comparison is performed based on architectures, fault tolerance and recovery in case of node failure.

The mining of these streams of data have challenges owing to the high-speed unbounded nature of them. The traditional data mining algorithms cannot be used in these cases and thus, new techniques need to be explored. This can be done either by extending and changing the existing systems to handle streaming data or by creating new streaming algorithms specifically for this purpose. Additionally, since the evaluation of performance is done continuously on the streams or on partially read streams, it is critical to use suitable performance measures. Thus, the paper focuses on challenges faced for mining streaming data, techniques and algorithms available and performance metrics.

II. RELATED WORK

The distributed stream processing systems are of high need in the industry and the fair amount of options available make the choice a difficult task. Thus, there have been studies reviewing and evaluating these systems. Hesse and Lorenz compare Apache Storm, Flink, Spark Streaming, and Samza platforms [7]. They describe only the architecture and its important elements.

Gradvohl et. al evaluate Millwheel, S4, Spark Streaming, and Storm systems [8] based on tolerance to failure. However, Landset et. al study big data processing tools [9], portraying the platforms' system architecture, nut this was mostly on tools that use MapReduce and thus are batch processing systems. Roberto Colucci et. al depict the performance of distributed stream processing systems for monitoring Signaling System number 7 (SS7) in a Global System for Mobile communications (GSM) machine-to-machine (M2M) application [10]. A comparative study of the Storm and Quasit, a prototype of University of Bologna is done which conveys that real-time processing of huge quantity of mobile application data is achievable using Storm.

Nabi et. al use an e-mail message processing application to perform a comparative study on IBM Infosphere and Apache Storm. The study concludes that Infosphere utilizes CPU better giving greater throughput, but Infosphere is not open-source. Lu et. al propose a benchmark [11]. They compare Apache Spark and Apache Storm in terms of latency and throughput but not on node failure. Also, Flink hasn't been included in the study. A comparison of architecturally different platforms like S, S4 and Event Stream Processor Esper. is performed by Dayarathna e Suzumura [12] along the lines of CPU utilization, memory usage and network costs. The manager/workers model is implemented in the S system, decentralized symmetric actor model is followed by S4, and finally Esper runs on Stream Processor. However, almost all of these systems are obsolete.

Ilaria Bartolini and Marco Patellain [13] use their novel RAM3S framework to compare of the three opensource big data stream processing platforms for identification of suspect people from huge video streams. Similar use cases may give the same results and performance. The comparison is again restricted to description of the architecture and its main elements to their use case. The same performance may change based on a different use-case. Thus, the use case is not generalized.

Gianpaolo Cugola and Alessandro Margara in [15] review certain Information Processing (IFP) Systems like CEP systems, databases and domain-specific systems and propose a generic model for IFP. They analyze the systems and conclude that each one fits a particular domain, stresses on different aspects and has unique mechanisms. However, this was mostly like a survey of methods available and the architecture, throughput and other such features weren't addressed sufficiently.

III. STREAM PROCESSING

3.1 Data Processing Approaches

There are three approaches to data processing: batch processing, micro-batch and stream processing. Batch processing technique gathers the incoming data for a period of time into groups called batches. Processing of the entire group is triggered at a later time based on a condition such as after certain minutes or after certain quantity of data has arrived. Hadoop or data warehouses work on this approach. However, this approach suffers from high latency and real-time applications need responses in microseconds. Micro-batch processing is similar to batch except the size of the batch is small or the processing is comparatively frequent. The data is still anyway processed as a batch. Spark streaming uses this approach. In these two techniques, it is easy to perform load balancing and fault tolerance. However, joining and splitting functions are quite difficult to perform on the entire batch.

In stream processing technique, the data streams are continuous and thus the data elements are processed as and when they arrive. However, most platforms offer 'window' feature based on a condition that works similar to micro-batch technique. The latency is this technique is lower and hence applicable for real-time scenarios. This makes fault tolerance and load balancing tasks challenging and calls for an event-driven architecture. Apache Storm and Apache Flink support this technique.

A Directed Acyclic Graph (DAG) is used to model stream processing. It consists of sources, sinks and processing nodes/elements. The data from stream source is channeled to the task processing nodes. These nodes analyze, evaluate and process the streams and pass them onto the sinks. The data from the sources in unbounded and continuous.

3.2 Fault Tolerance and Recovery

The stream processing systems cannot compromise on availability and need to ensure that the data is processed at any cost. Thus, recovery after failure is important for these systems. Since the systems are distributed, failure can occur on any cluster node, network or the software. Latency is not acceptable in case of stream processing applications and failure means data loss. Thus, there are certain message delivery mechanism followed, namely at least once, at most once and exactly once, which are the levels of guarantee given by the system regarding the data handled by it. A node can pass the data to a different node in case of a failure without any data loss. The simplest mechanism is at most where the message handed to the mechanism is delivered either zero or one times. There is no guarantee of error recovery and the message may be lost. In case of at least once mechanism, multiple attempts are made in delivering the message to ensure that at least one is successful. This can lead to message duplication. In exactly once mechanism, it is ensured that only one message is delivered to the recipient by acknowledging the receipt.

For a batch processing system like Hadoop, the requirements are to perform correct calculation without losing any data even in case of failure and thus latency is compromised. For a stream processing system, the expectation is quick recovery in case of a failure with minimal effect on system functioning without compromising latency. Thus, recovery mechanisms are important for stream processing and these range from recovery without any loss of data to recovery with minimal loss of data. The overall system functioning may or may not be affected. There are three types, namely precise, rollback and gap recovery. In precise and rollback schemes, there is higher latency but in case of the latter, there is an additional effect on system functioning, like duplication of data. The one with the least guarantee for processing and with a loss of data is gap recovery. The conventional mechanisms use active, passive, upstream backup nodes or amnesia. The latter provides gap recovery with negligible overhead and the first three can provide both rollback and precise recovery.

3.3 Challenges faced in mining of streams of data

Since the streaming data is continuous and unbounded, there are several challenges and issues in order to analyze and extract inferences from this data. The following are the challenges faced for mining high-speed infinite streaming data:

- Memory requirements are huge for the data as it is unbounded i.e.; essentially infinite and high-speed. Thus, mechanisms have to developed to address the same with lesser memory requirements.

- The inferences from the data are expected to be provided in real-time and thus the techniques used for the same can perform a single scan of the streams to provide accurate results. This reduces the time to compute is comparatively lesser.
- The most important issue is the evolving nature of these streams of data. For example, during classification, the features of some data which is being trained may change over time. For clustering, the number of clusters may undergo a change gradually. Also, noise may be added to the data over the network. Thus, effective mechanisms are required for mining this kind of data.

3.4 Classification Algorithms

Classification is the process of predicting the class label of an unknown data instance based on the model constructed from the training data. In streaming algorithms, model construction and testing go hand in hand as it is not possible to batch the data. Some of the algorithms are:

- Streaming Ensemble Algorithm – It is basically a fast algorithm for large-scale or streaming data that classifies the data while building a single decision tree on all the data. This needs approximately constant memory, is robust and adjusts quickly to concept drift.
- Weighted Classifier Ensemble – Uses ensemble of weighted classifiers built on data elements and deals well with drifts.
- On Demand Classifier – Uses micro clustering technique and windows dynamically to ensure greater performance.
- Evolving Naïve Bayes – An extended Naïve Bayes algorithm that learns from evolving data streams.
- Adaptive Nearest Neighbour Classification Algorithm – An incremental algorithm that adaptively searches for nearest neighbours by multi resolution representation of data. It facilitates low model cost.

3.5 Clustering Algorithms

Clustering segregates the incoming data elements into the different clusters, which in turn may change gradually. Thus, time-span is also considered for this process.

- Stream K Means: Clusters over the complete data stream and thus doesn't support evolving streams.
- E-stream: Uses five types of evolutions: merge, split, self-evolution, appearance and disappearance to detect changes while clustering.
- ClusTree: A hierarchical data structure that is self-adaptive is maintained offline. It also tags an age to each data element to ensure that only the latest elements contribute to changes.
- Hyper-Ellipsoidal Clustering for Evolving data Stream (HECES): It uses ellipsoid shaped cluster merging instead of window expansion and contraction to achieve faster results on varying density clusters. Thus, conforms to sliding window mechanism to handle evolving data.
- Correlation Clustering in Data Streams: It provides a set of space and time efficient techniques for convex programming that solves the problems of correlation clustering that arise in dynamic data streams.

3.6 Metrics of evaluation

There are some metrics that can be used for evaluating the performance of these algorithms based on their type. For classification, there are the following metrics:

- Kappa statistics- Evaluates performance for an imbalance stream case and the value is directly proportional to the performance.
- Temporal-Kappa Statistics-Evaluate performance for temporal dependent data streams. Worse performances are indicated by negative values.

For clustering, the metrics mentioned below are used:

- Completeness: Checks the validity of clustering against data elements of same class. The value is directly proportional to performance.
- Purity: Evaluates the clusters by checking outliers in the clusters. Greater value implies better performance.
- SSQ: Measures clusters cohesiveness. Lower value implies better performance.
- Silhouette Coefficient: Assess compactness as well as separation of clusters. The value is directly proportional to performance.

IV. STREAM PROCESSING PLATFORMS

4.1 Apache Samza

Apache Samza [11] was developed by Linkedin and then taken up by Apache Software Foundation. The stream consists messages of same type that are immutable. The messages can be received by multiple consumers, can be appended or deleted from the streams. The brokering layer persists the streams. The processing units, tasks, work on a message stream and output result streams. This is called a job. In order to achieve parallelism, the stream may be partitioned into ordered sequence and distributed among the tasks for execution. The resource allocator and scheduler is Apache Yarn, which manages the cluster. [20] The brokering layer is handled by Apache Kafka. APIs are provided for running the jobs on the cluster.

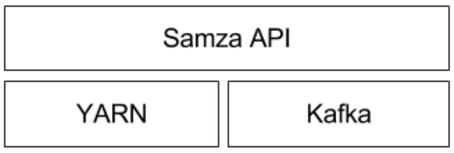

Figure 1: Three layers in Samza architecture
Retrieved from: https://samza.apache.org/learn/documentation/0.14/introduction/architecture.html

In Samza architecture, the streaming layer is Kafka, the execution layer is YARN and the processing layer is Samza API as shown in Figure 1. The messages written to the brokering layer are persisted to the file store by Kafka, where they are stored for the configured duration. The downstream tasks can consume the messages at any point within that time frame. The tasks are written in Java which specifies the processing methodology. The actual job is configured using a properties file. The compiled java code and the properties file are submitted to the cluster.

Samza provide at least once message delivery guarantees. When a node fails, the node that takes over reads from the upstream backup based on the offset that the failed node was using. This recovery works at task level, however if a broker fails, data is lost. Hence, the file system is used to handle such failures by backing up the data.

4.2 Apache S4

S4 [12] stands for Simple Scalable Streaming System and it was developed by Yahoo and donated to Apache Software Foundation. The processing model is inspired by MapReduce and uses key based programming model as in MapReduce. Processing Elements (PE) perform the core computations and S4 creates a network of these Pes as a DAG at runtime as an instance for each job submitted. The PE's code, configuration, events consumed, key attributes for the events and value of the keyed attributes are used to identify a runtime instance.

The PEs are allocated to Processing Nodes which behave as their logical hosts. The key attribute values of the events submitted to the Processing Nodes are hashed and assigned to the PEs. A communication layer is dedicated to enable the coordination between these nodes and the messaging system. Zookeeper is used for cluster management and coordination. S4 jobs are written using Java. The configuration for a job is done through the Spring Framework based XML configuration. A job configuration consists of the definition of the included PEs and how they are configured.

S4 uses Gap recovery technique and provides no message delivery guarantees. Zookeeper identifies a failed node and the tasks of that node are distributed across the cluster. To enable this, snapshots of processing node state are saved periodically.

4.3 Apache Storm

Apache Storm [8] is a real-time stream processor, written in Java and Clojure. Stream is called tuples and has the data and an identifier. Spouts are input nodes and the processing nodes are bolts. The jobs are topologies (similar to MapReduce in Hadoop) of DAGs. The link between nodes in the graph is defined by grouping type, which enable the programmer to define flow in the graph. There are eight grouping types and the main ones are: shuffle, field, and all grouping. In shuffle grouping, all bolt instances receive the stream. In field grouping, each bolt instance is responsible for all samples with the same key specified in the tuple. Finally, in all grouping, samples are sent to all parallel instances.

The manager node, Nimbus, receives a user-defined topology and coordinates the instantiation. Each Worker node runs on a Java Virtual Machine to execute one or more tasks, called processes. The Supervisors monitor the processes of each topology and inform the state to Nimbus using the Heartbeat protocol. Zookeeper ensures coordination between Nimbus and Supervisors and is responsible for state management as Nimbus and Supervisors are stateless. The architecture is shown in Figure 2.

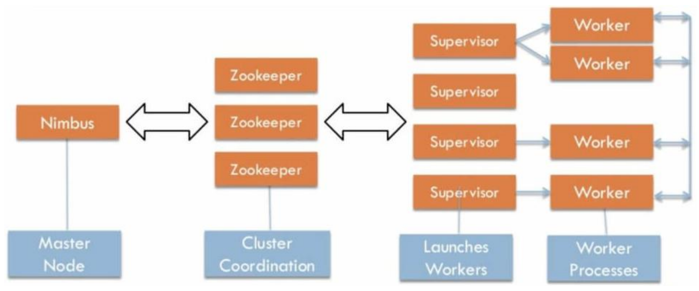

Figure 2: Storm architecture
Retrieved from: https://andyleonard.blog/2018/01/apache-storm-stream-processing/

Storm also relies on upstream backup in case of fault tolerance. Spouts hold messages in output queues until an acknowledgement is received. It guarantees at least once message delivery by resending the message if an acknowledgement doesn't arrive within a specified time. Node failure is detected by Nimbus by using heartbeat messages sent by Supervisors. The message and node failures are handled separately and thus makes the system robust.

4.4 Apache Flink

Apache Flink [5] is a hybrid processing platform, supporting both stream and batch processing. The Flink applications translate into DAGs specifying sources and sinks of streams. The Flink architecture has job manager and task manager. The job manager manages the submitted jobs by assigning them to task managers and parallelizing the task. The stream abstraction is DataStream. They can take input from databases, Kafka topics etc. by providing connectors to them. There are several operators that can be used in processing of data like map, filter, co-process. It is possible to set a different parallelism for each operator. This is shown in Figure 3.

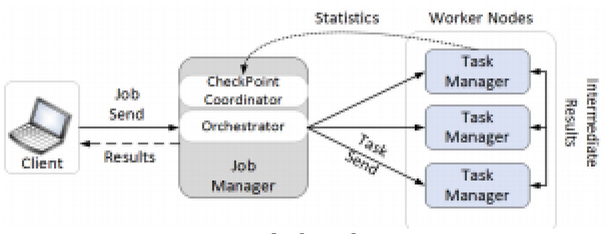

Figure 3: Flink architecture
Retrieved from: A Performance Comparison of Open-Source Stream Processing Platforms - Scientific Figure on ResearchGate. Available from: https://www.researchgate.net/figure/Architecture-of-Flink-system-The-Job-manager-receives-jobs-from-clients-divides-the_fig3_309322789.

Flink ensures that fault tolerance is achieved by using snapshots for checkpointing. Barriers pass through the graph along with the stream and indicate beginning of a checkpoint. Barriers trigger new snapshots of the state when they pass through operators. When an operator receives a barrier, it stores the status of the corresponding stream snapshot and sends the checkpoint coordinator to the job manager. In case of network or node failure, Flink stops and restarts execution from the recent checkpoint. Thus, exactly once message delivery is ensured.

4.5 Apache Spark

Streaming Spark [6] is a platform for distributed data processing, written in Java and Scala. It supports stream processing as micro-batch processing using Spark Streaming which is a library that runs on spark Engine. The stream abstraction is called Discrete Stream (D-Stream) which is a set of short, stateless, deterministic tasks. When a stream enters Spark, it divides data into micro-batches, which are the input data of the Distributed Resilient Dataset (RDD), the main class in Spark Engine, stored in memory.

Spark Context can run in environments of resource managers like Mesos, YARN or as standalone. The master or driver program runs the jobs on the cluster as independent processes on task managers as shown in Figure 4. The mechanism described in Storm, in which each worker process runs within a topology, can be applied to Spark, where applications or jobs are equivalent to topologies. A disadvantage of this concept in Spark is the messages exchange between different programs, which is only done indirectly such as writing data to a file, worsen the latency that could be around seconds in applications of several operations.

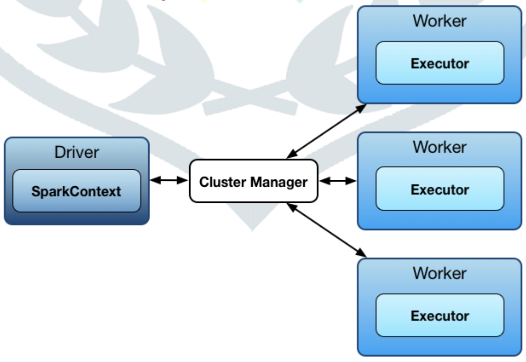

Figure 4: Spark architecture
Retrieved from: The Spark Internals
Available from: https://jaceklaskowski.gitbooks.io/mastering-apache-spark/spark-architecture.html

The Spark has exactly once delivery guarantee. The state management ensures that snapshots of RDD are periodically taken. Thus, in case of a fault the micro-batch is then distributed onto other nodes in the cluster, thus ensuring parallel recovery. However, micro-batch processing has limitations as it takes longer in downstream operations.

V. CONCLUSION AND FUTURE WORK

This paper highlights the current big data trends of stream processing. It explains the pitching applications of stream processing in various fields. It further discusses the related work in the field and presents the need for the survey. The data processing approaches highlight batch, micro-batch and stream processing and the differences between them. The issues and challenges faced because of the unbounded nature of streaming data are discussed and the techniques, measures to evaluate the same.

Finally, the stream processing platforms are discussed and compared. It can be seen that the platforms designed for stream processing like Flink and Storm perform better than Spark which uses micro-batching.

This paper highlights the importance of fault tolerance and the ways used to recover from the same. The stream processing platforms are compared for the same criteria. Thus, the paper presents an overall survey with respect to stream processing. The future work for the next phase is to implement the stream processing techniques on the platforms and perform the comparison of the metrics, fault tolerance and the recovery mechanisms.

REFERENCES

[1] M. Stonebraker, U. C¸ etintemel, and S. Zdonik, "The 8 requirements of real-time stream processing," ACM SIGMOD Record, vol. 34, no. 4, pp. 42–47, 2005.

[2] P. Clay, "A modern threat response framework," Network Security, vol. 2015, no. 4, pp. 5–10, 2015.

[3] A. Toshniwal, S. Taneja, A. Shukla, K. Ramasamy, J. M. Patel, S. Kulkarni, J. Jackson, K. Gade, M. Fu, J. Donham, N. Bhagat, S. Mittal, and D. Ryaboy, "Storm@twitter," in ACM SIGMOD International Conference on Management of Data. ACM, 2014, pp. 147–156.

[4] Sanket Chintapalli, Derek Dagit, Bobby Evans, Reza Farivar, Thomas Graves, "Benchmarking Streaming Computation Engines: Storm, Flink and Spark Streaming", *IEEE International Parallel and Distributed Processing Symposium Workshops*, 2016, pp. 1789-1792.

[5] Marco Aldinucci, Maurizio Drocco, Claudia Misale and Guy Tremblay, "Languages and Frameworks for Big Data Analysis", *Encyclopedia of Big Data Technologies*, S. Sakr and A. Zomaya, Eds., Springer, 2018, pp. 1-16.

[6] Salloum, S., Dautov, R., Chen, X., Peng, P. and Huang, J, "Big data analytics on Apache Spark", *International Journal of Data Science and Analytics*, Vol. 1 Issue 3-4, 2016, pp. 145-164.

[7] G. Hesse and M. Lorenz, "Conceptual survey on data stream processing systems," in IEEE 21st International Conference on Parallel and Distributed Systems, 2015, pp. 797–802.

[8] A. L. S. Gradvohl, H. Senger, L. Arantes, and P. Sens, "Comparing distributed online stream processing systems considering fault tolerance issues," Journal of Emerging Technologies in Web Intelligence, vol. 6, no. 2, pp. 174–179, 2014.

[9] 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," Journal of Big Data, vol. 2, no. 1, pp. 1–36, 2015.

[10] R. Coluccio, G. Ghidini, A. Reale, D. Levine, P. Bellavista, S. P. Emmons, and J. O. Smith, "Online stream processing of machineto-machine communications traffic: A platform comparison," in IEEE Symposium on Computers and Communication (ISCC), June 2014, pp. 1–7.

[11] R. Lu, G. Wu, B. Xie, and J. Hu, "Stream bench: Towards benchmarking modern distributed stream computing frameworks," in IEEE/ACM 7th International Conference on Utility and Cloud Computing, 2014, pp. 69–78.

[12] M. Dayarathna and T. Suzumura, "A performance analysis of system S, S4, and Esper via two level benchmarking," in Quantitative Evaluation of Systems. Springer, 2013, pp. 225–240.

[13] Ilaria Bartolini and Marco Patella, "Comparing Performances of Big Data Stream Processing Platforms with RAM3S," DISI - Alma Mater Studiorum, Universit`a di Bologna, 2015, pp.1-8.

[14] Bakshi Rohit Prasad and Sonali Agarwal, "Stream Data Mining: Platforms, Algorithms, Performance Evaluators and Research Trends," International journal of Database Theory and Applications, Vol. 9, No. 9, 2016, pp 201-208.

[15] Cugola, Gianpaolo and Alessandro Margara, "Processing Flows of Information," ACM Coputing Surveys, Vol44. No.3, 2012, pp. 1-62.

[16] Martin Andreoni Lopez, Antonio Gonzalez Pastana Lobato, Otto Carlos M. B. Duarte, "A performance comparison of Open-source stream processing platforms," 2016 IEEE Global communications Conference (GLOBECOM), 2016, pp. 1-6.

[17] P. Carbone, G. Fora, S. Ewen, S. Haridi, and K. Tzoumas, "Lightweight ´ asynchronous snapshots for distributed dataflows," Computing Research Repository (CoRR), vol. abs/1506.08603, 2015.

[18] M. Zaharia, T. Das, H. Li, T. Hunter, S. Shenker, and I. Stoica, "Discretized streams: Fault-tolerant streaming computation at scale," in XXIV ACM Symposium on Operating Systems Principles. ACM, 2013, pp. 423–438.

[19] A. Lobato, M. A. Lopez, and O. C. M. B. Duarte, “An accurate threat detection system through real-time stream processing,” Grupo de Teleinformatica e Automac¸ ´ ao (GTA), Univeridade Federal do Rio de ˜ Janeiro (UFRJ), Tech. Rep. GTA-16-08, 2016.

[20] D. Carney, U. C¸ etintemel, M. Cherniack, C. Convey, S. Lee, G. Seidman, M. Stonebraker, N. Tatbul, and S. Zdonik, “Monitoring streams: A new class of data management applications,” in 28th International Conference on Very Large Data Bases, 2002, pp. 215–226.