Certain Investigation and Supervision on Big Data Live To Stream and Predicts Desirable Data from Various Resources with the Interior of Big Data

Sasikumar.P[1] , Kalaivani.K[2]

[1]Research Scholar, Department of Computer Science and Engineering, VISTAS, Pallavaram, Chennai, India

npsasikumarmtech@gmail.com

[2] Assistant Professor, Department of Computer Science and Engineering, VISTAS, Pallavaram, Chennai, India

kalai.se@velsuniv.ac.n

ABSTRACT

Real-time streaming data analysis is becoming the quickest and most effective approach to learning something from what is happening right now. This allows businesses to respond quickly when issues arise or to see emerging trends that can help them perform better. Due to the big data's unique dynamic properties, it was impossible for conventional data mining tools and techniques to be deployed directly on large data streams. This study presents a thorough evaluation of big data stream analysis and parallel mining, using the rigorous methodology to examine the developments in big data stream tools and technologies. The study discovered that there is still space for more research on huge data streams and technologies, as well as scalability, privacy, and load balancing challenges. Furthermore, this paper also discovered that despite major research efforts being focused on real-time analysis of Big data streams in parallel mining. Only a small number of big data streaming technologies are capable of performing all batch, iterative and streaming; at the moment, it appears that no big data tool or technology delivers all the essential capabilities needed, and a common benchmark dataset for big data streaming analytics is not yet commonly used. It was proposed that research efforts should concentrate on creating scalable framework and methods that will enable data stream computing mode, efficient allocation of resources strategy, and parallel processing challenges in order to deal with the ever-increasing volume and complexity of data. For real-time prediction and analysis, evolving data stream technologies are emerging as an affordable, environmentally friendly option. The problems that the field will have to overcome over the coming years are discussed, along with present and upcoming trends in mining evolving data sources.

Keywords: *Stream computing, big data stream analysis, live streaming, and parallel mining*

I. INTRODUCTION

As the fourth paradigm of science, data-intensive science entails gathering, organizing, analyzing, and disseminating data in order to advance scientific knowledge. In addition to the ability to analyze massive quantities of data, it is necessary to support real-time judgments based on sporadic data availability due to the widespread use of sensor technologies as a tool to gather physical measurements. These use-cases have been addressed by current distributed technologies like Hadoop, but they are frequently inappropriate for use-cases involving non-deterministically available data or dynamic real-time data [1]. There are many real-time data processing platforms now being developed and used in both industry and academics.

Huge amounts of information may be continuously produced by a data stream system. Consider a multi-sensor system with 10,000 sensors that would provide measurements once every second to provide an example. The constant delivery of data items in multiple, quick, time-varying, and potentially limitless streams creates new challenges and research problems in terms of data storage, management, and processing. In fact, just saving the incoming data in a conventional database management system and performing actions on it afterward is typically not practical. In order to ensure that results are current and that queries may be replied to with little to no delay, stream data must be processed online.

Despite the wide range of data domains and business logic, the majority of Stream Processing Systems have a generic data processing pipeline [2]. The Data Stream Processing System is made up of the following layers: I a data stream ingestion layer on the front-end, which is in charge of acknowledging amounts of data into the Streaming Data Processing System; (ii) a streaming data processing layer; (iii) a storage layer, which stores, indexes, and retains the data and the created knowledge; (iv) a resource management layer, which regulates and collaborates the processes of distributed computing and storage resources; and (v) a layer for output that sends the knowledge and data stream towards other systems or visualisation tools.A general Data Stream Processing System [4] architecture is shown in Figure. 1.

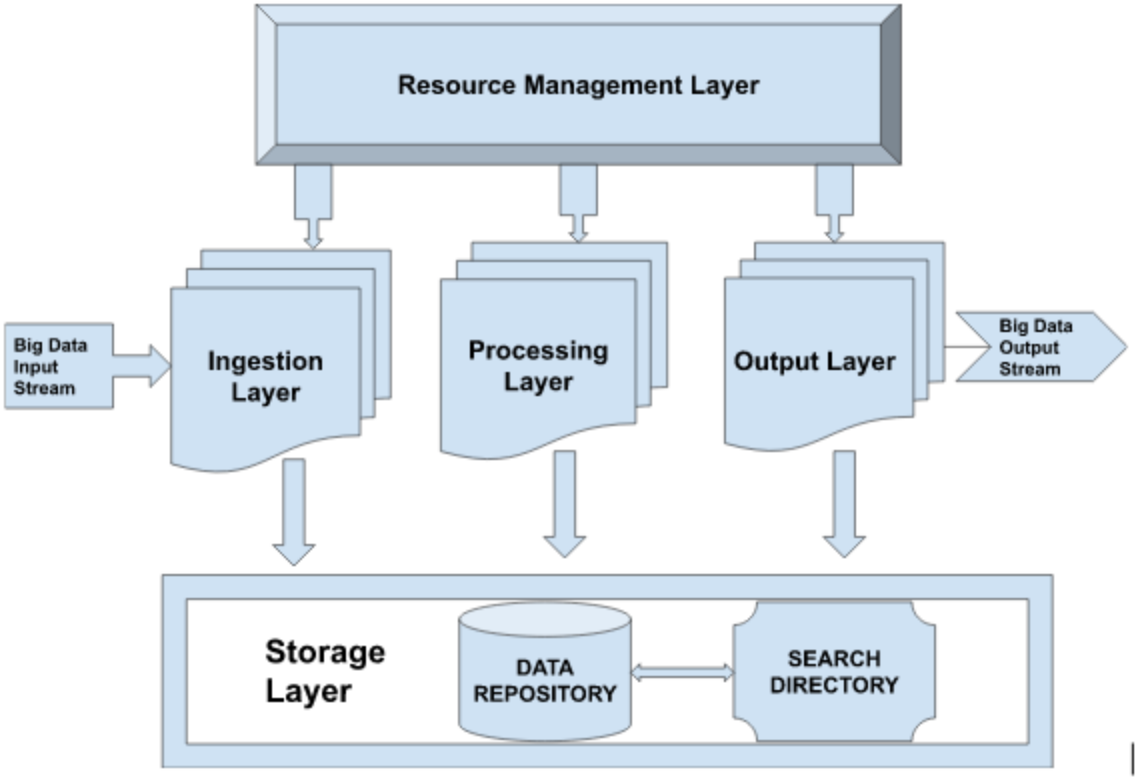

Figure 1: Data Stream Processing System architecture.

For instance, in smart cities, a variety of sensors are installed in various locations (e.g., water lines, power lines, buses, railways, and traffic signals) [5]. These sensors include GPS, weather devices, smart cards for public transit, and traffic cameras. Large amounts of data are gathered from these sensors. It is crucial to discover hidden and significant information within the large stream/storage of data in order to comprehend such a volume of data. Another representative source of huge data that needs in-the-moment processing and outcomes is social media [6]. In actuality, a sizable amount of data is generated quickly and continually from a variety of Internet apps and Web sites. Examples include social networks like Facebook and Twitter, online mobile video and image sharing services like Instagram, Youtube, and Flickr, as well as networks for businesses (eg.LinkedLn). Figure 2 shows the phases of big data processing.

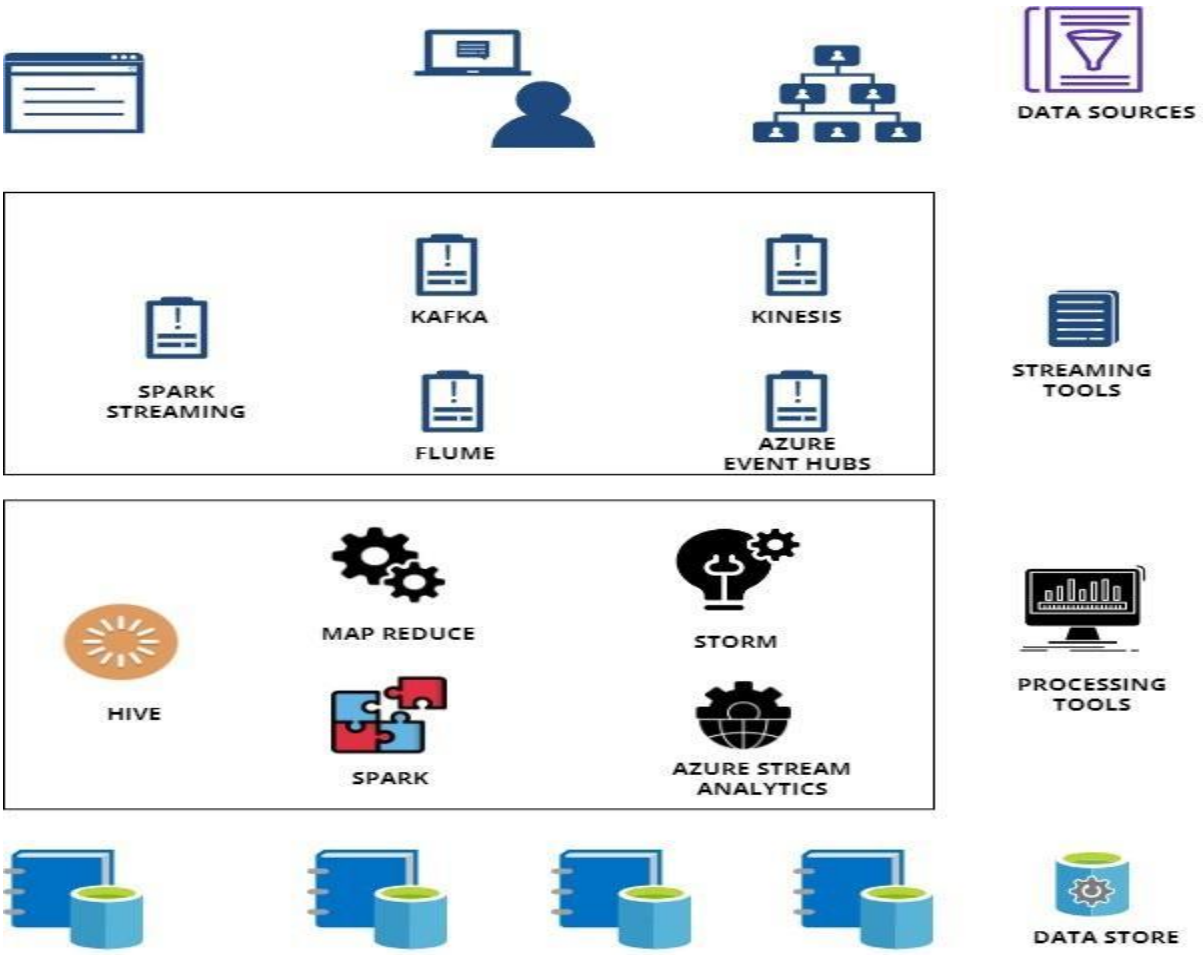

Figure 2 : Phases of Big data Processing

Certain tasks, such as social network analysis and link prediction, can be successfully completed by utilising in-stream frameworks that offer iterative learning and processing capabilities. The importance of the real-world scenarios that were addressed makes it challenging to find the right architecture for big stream-oriented applications.

The popularity of data mining has led to the development of new algorithms that can analyse large, infinite data flows that come from a variety of sources, such as sensors, network connections, call centre records, social media posts, ATM and credit card transactions, web searches, etc. Large data streams, that are continuous series of data generated at a high rate of generation, are produced by all of these sources. [2].

Each component of a stream consists of a pair of coordinates (Xi,Ti) numbers, where Xi is a dimensional vector and Ti is the time stamp. The element's time stamp indicates when it happened [3]. The process of continuously mining these kinds of data streams involves taking knowledge and priceless insights out of limitless and quick data signals. Using machine learning

techniques like classification, clustering, regression, etc., the basic objective is to forecast the class or the value of the new items [2].

Table 1. Real-life applications of Data Stream Mining

Domain Areas	Type of Streaming Data	Need for Stream Processing
Healthcare	Medical records	Diagnosis of acute diseases and providing suggestions for patient health monitoring
Biological Data Analysis	Genomic and Proteomic data	Discovering structural patterns, Comparative analysis of Genetic data
Financial Data Analysis	Multidimensional Banking Data	Money laundering detection, Targeted Marketing, Prediction of Loan Payment
Retail Industry	Sales Data	Retention of Customers, Recommendation of products
Telecommunication Industry	Satellite signals	Sequential Pattern Analysis, Analysis of web data transmission
Intrusion Detection	Network data	Correlation analysis for network administration

There are numerous frameworks that can process data streams, therefore picking the right one that satisfies the problem requirements is crucial. Some well-known processing frameworks for mining streaming data include Hadoop [8], Spark [9], Storm [10], Flink [11], Massive Online Analysis (MOA) [3], and Waikato Environment for Knowledge Analysis (WEKA) [7].

Table 2. Traditional Mining Approach vs Data Stream Mining Approach

Features	Traditional Mining Approach	Data Stream Mining Approach
Passes required	Many	One
Time is taken for processing	Limitless	Finite
Memory consumed	Limitless	Finite
Distributed in Nature	No	Yes
Type of Data used	Bounded data	Continual steady data
Evolution of Data	Stable	Flexible

Observation sequence	Independent	Dependent
Stability of models	Fixed	Continually updated
Labeling of records	Low cost	High cost
Results obtained	Precise	Approximate

The article's remaining sections are organised as follows: In the section titled "Background and Related Study," which also gives data on stream computing, big data stream analysis, parallel mining, and the important challenges associated in it, a study of big data streaming analytics is included. Big data stream analysis and parallel mining are highlighted in the section under "Limitation of the review." "Conclusion and further work," that addresses the study's shortcomings, comes at the end of the research.

2. Background and Related Study

Real-time processing is required by more than 70% of respondents, according to a 2013 industry survey on big data technology utilization in European companies [11]. Through the quick growth of contemporary Data Stream Processing System technologies, the open source software community has undoubtedly responded since that time.

2.1 Stream computing

Stream computing is the real-time processing of enormous amounts of data that are created at high velocity and from several sources with minimal latency. It is a new paradigm that is necessary due to new data-generating scenarios, such as the widespread usage of location tracking, handheld platforms, and sensors [12]. It can be used to process data flowing at high speeds from real-time sources such the Internet of Things, sensors, market data, mobile devices, and click streams. As contrast to batch computing, wherein data are first kept before being examined, data are analysed as soon as they enter in a stream to provide results. Scalable computing platforms and parallel architectures are absolutely necessary [13]. Organizations can use stream computing to study and react in actual to fast evolving data. The frameworks for stream processing include Storm, S4, Kafka, and Spark. [14-16].

Samza[35] and Storm have a lot in common in terms of functionality. Samza is built on mature components like YARN and Kafka, to which much fundamental functionalityis unloaded, even if it lacks the maturity and adoption rates of projects like Storm. Samza handles data streams by running predefined tasks that apply any specified procedure to the streams of data. Kafka, which can be utilized as an output location for jobs for additional pipeline-style processing, is the source of streams in Samza. Table 1 presents the key differences between both the batch processing and stream processing approaches listed.

Table 3: Batch Processingvs Stream Processing

Characteristics	**Batch Processing**	**Stream processing**
Input Data	Data blocks	Streams of Data
Size	Finite	Infinite
Hardware employed	Numerous Processors	Bounded memory
Storage	Data Repository	Less/No storage in memory
Processing	Multi-rounds for processing	Single-round for processing
Time taken	Longer	Shorter

2.2 Big data stream analysis

The core of big data streaming analytics is the requirement to analyze and react to real-time streaming data via continuous queries, allowing for continuous on-the-fly analysis within the stream.

According to the streaming analytics concept, the data processing node processes each data tuple that is received. Because of the extremely high data rates of external feeds, such processing often involves a single pass and comprises tasks like deleting duplicates, filling in gaps in data, data normalization, parsing, and feature extraction. This node is activated when a new tuple enters, and it deletes any tuples that are older than the time period indicated in the sliding window. A common type of window used during stream computing is the sliding window, which only stores the most recent tuples up to the time indicated in the window. Logical containers for receiving data tuples are referred to as windows. It specifies when the processing of data is triggered as well as the frequent data is updated in the container [4].

Gama, J [18] compared platforms for large data processing using measures like scalability, fault tolerance, and data I/O rate, although their work was mostly focused on hardware platforms and only a few software frameworks like Hadoop [7] and Spark [9], which are used for batch processing. Big data batch processing was a focus of distributed processing in previous years.

The Storm project [17] is a well-known example of a Data Stream Processing System that was created apart from Hadoop and is rapidly gaining popularity and a growing user base. Initially, a BackType engineering team under the direction of Nathan Marz created Storm. The storm is described by Toshniwal et al. [12] as "a real-time distributed stream data processing engine" that "powers the real-time stream data management operations that are required to provide Twitter services" [17].

Through Spark's fundamental data abstraction, the resilient distributed dataset (RDD), and its innovative approach to in-memory computation. Data streaming can be processed thanks to Spark Streaming [8], which is based on the Spark engine. Samza is a relatively recent framework for real-time large data processing that was initially created internally at LinkedIn and has now been made available through the Apache Software Foundation [9].

Table 4. Tools and technologies for big data stream analysis

S.No	Author	Tools and technology
1.	Krawcyz et.al.,	Spark Streaming
2.	Nguyen, H.L.et.al.	Apache storm
3.	Vanathi R, Gokalp MO, et.al. Fernandez-Rodrigues JY et.al.	Kafka
4.	Inoubli W, Liao X et.al.	Apache Samza
5.	Ren X et.al.	SAMOA
6.	Bifet et.al.	Apache Flink
7.	Tyler Akidau et.al.	Google Mill Wheel

3. Issues in big data stream analysis

Big data streams have a number of difficulties that must be overcome, including sustainability, connectivity, flexibility, reliability, coherence, diversity and incomplete data, task scheduling, privacy concerns, and correctness [12].

Table 5: Comparison among tools and technologies used in Big data stream analysis

Tools	Model	Fault tolerance	Latency	Through put	|Reliability	OS	Database support	Application
Spark Streaming	Streaming	RDD based checkpoint, parallel recovery, and replication	Low	High	Exactly once	Windows	Kafke	Streaming, fog computing, multimedia analysis
Apache storm	Streaming	Check-point. Replication, recovery, back-up	Very Low	Low	At least once	Windows	HBase, Cassandra, HIVE	IoT, Streaming, multimedia analysis
Apache Samza	Streaming, batch processing	Check point	Very Low	High	Atleast once	Windows	Kafke, HDF5	Reprocessing, invalidating cache
SAMOA	Streaming,	Upstream backup	Low	High	Exactly once	Linux	Hive, Cassandra, HBase	Classification, clustering, regression
Apache Flink	Streaming,	Stream replay	Very low	high	Exactly once	Windows, Liunx	Kafka, Flume, HBase	e-commerce, Network or sensor monitoring
Splunk stream	Streaming,	Check point, Upstream backup	Low	High	Exactly once	Linux	BigTable, Spanner	Anomaly detection, health monitoring

Table 5. An overview of scheduling approaches in stream processing big data frameworks.

System	Execution Model	Schedule Decisions	Tools	Comparedwith	Evaluation Metrics
Spark Streaming	Micro batches	Offline	FIFO	Storm	Better: Throughput : 5x-6x
Storm	Operator based	Offline	Round Robin	Spark Streaming	Better:Latency:50x
Flink	Operator based	Offline	-	Spark Streaming, Storm	Better:Throughput Less:Memory utilization
Liao et al.	Micro-batches	Online	Timer	Spark Streaming	Better:Latency
T-Storm	Operator based	Online	Load monitors, ML	Storm	Better:Throughput Less: Number of work nodes
Safaei	Operator based	Online	System's response time	Storm	Better: Latency More: Memory Usage

4. Parallel Computing

The Parallel Random Forest (PRF) algorithm on the Apache Spark platform [34] has been introduced by Jianguo Chen et al. [4]. An effective data mining algorithm for big data was the Random Forest algorithm. On a hybrid parallel method, the PRF algorithm was built. The innovative Parallel Regular Expression Matching algorithm (PaREM) was developed by J. Chen et al.

4.1 Parallel Paradigms

Big data mining has two main difficulties. First, the speed at which the data are generated exceeds just what the machine can handle in its available memory. To calculate the patterns based on such a vast amount of data comes second. Therefore, two resources—data access and its computation—would be needed for any framework for large data mining. When working with small data sets, mining is simple to execute on a single computer because all of the processing may be done in the machine's main memory. Even if the datasets fit within the main memory, it is still feasible that the intermediate processing won't be able to accommodate memory. For this type of computing operation, more single computing node or cluster is needed. Thus, one of the most effective ways to solve the issue of mining from enormous datasets is through parallel processing. However, creating an effective algorithm for parallel processing is similarly difficult

since the method must address many issues, including data security, load balancing among nodes, scalability, the division of work, communication costs, synchronization, and error handling. This section talks about a few parallel and distributed frameworks.

Grid Computing: It may be described as a computer network where a computer's resources—such as processing speed, memory, and data storage—are shared with other computers inside the company in order to accomplish a common objective. Each computer may have a separate mission or job to carry out while the computers are working in heterogeneity. The entire system is built to operate in parallel with great performance, especially when all of the components are not nearby.

Multi-core Computing: In the multi-core computing paradigm, two or more separate processors are housed on a single silicon chip. Due to pipelining and multithreading, this method achieves a higher level of parallelism than grid computing. While one instruction is being executed in one stage of the pipeline, another instruction is being executed in a different stage. High scalability is supported by multi-core computing, albeit at the expense of making chip fabrication and debugging more challenging.

Graphics Processing Units: A programmable computer processor known as a graphics processing unit, or GPU, is capable of quickly and heavily parallelizing calculations. They were initially developed to give screens a 3D visual impression. Initially, the CPU handled all of the computations, but when demands for computations increased, GPU stepped in and took over. Recently, appliances that use GPU and CPU for image processing, parallel graph evaluation, and other deep learning approaches have been proposed.

Hadoop and MapReduce: The necessity for quick and effective processing and the explosion of data created the necessary conditions for the creation of distributed algorithms. MapReduce is a programming methodology that has been used for the past few years to create and implement these algorithms on the distributed platform Apache Hadoop. Hadoop is a collection of free software tools for processing large amounts of data across a network of computers. It may be supported by a local data center or through the use of clouds. MapReduce can be used to create

applications that process massive volumes of data over several nodes or clusters. It is a distributed computing programming model that is based on Java. MapReduce handles the two crucial functions of Map and Reduce. It divides the input data into separate, autonomous chunks of data with some replicates, which the mapper then processes in parallel. The output is created as key-value pair pairs that are sorted before being supplied as input to the reducer. Based on the keys, the reducer reduces the total amount of data. The MapReduceprogram serves as a general scheduler and monitoring. Additionally, it restarts failed jobs without interfering with ongoing work. Additionally, it accomplishes load balancing by reassigning incomplete work from malfunctioning or out-of-order nodes to other vacant nodes. Another part of Hadoop called HDFS or Hadoop Distributed File System is used to reliably and distributedly store massive datasets. Hadoop has some restrictions, though. For some of the issues, the Hadoop key-value paradigm might not be the best solution. Additionally, the discs are used for both reading and writing operations. Every iteration must process data from the disc once more, which is an expensive procedure and limits Hadoop's flexibility and functionality. Figure 3 describes the framework of data processing in parallel mining.

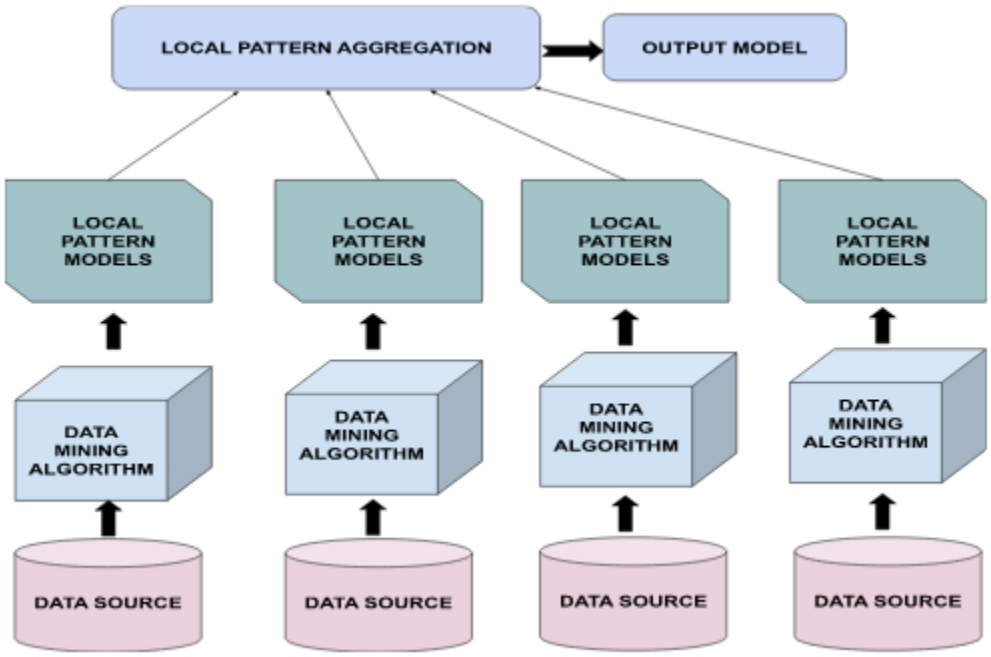

Figure 3: Data Processing in Parallel mining

Spark: The most popular distributed computing framework is now Apache Spark. With its in-memory compute feature, it surpasses Hadoop as a large-scale data processing integrated analytics engine. Large data sets make it challenging to read from the disc for each iteration,

which strengthens the case for Spark's development and use over Hadoop. Even for massive clusters, in-memory processing is performed at every iteration in RDD. RDD is a fault-tolerant data set made up of read-only data elements that are dispersed over many clusters. Spark allows for independent Map and Reduce operations, which means that one is not required to come after the other. Spark is hence more adaptable than Hadoop. Spark is depicted in Figure 4 describes the framework of Apache spark architecture

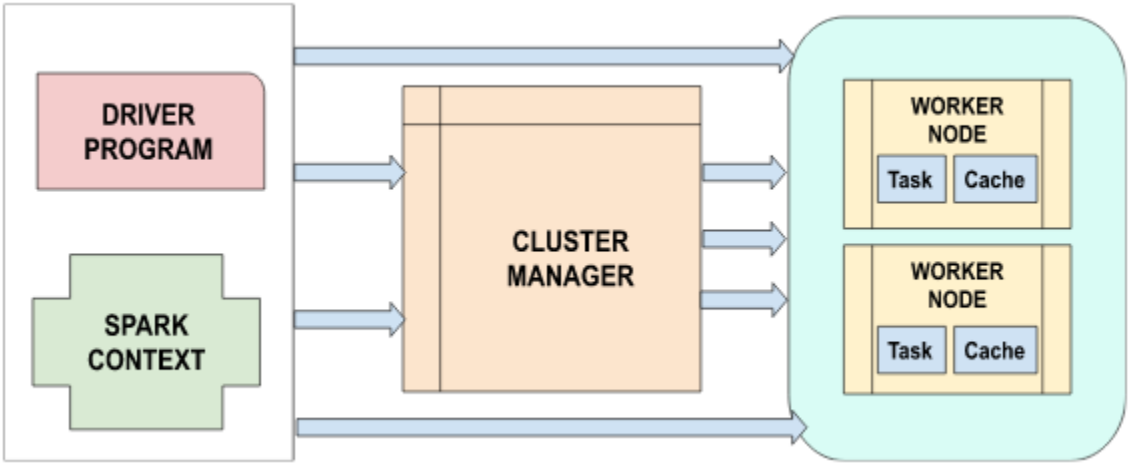

Figure : Apache Spark Architecture

5. Assessment Criteria

The above-mentioned frameworks and architectures are examined in light of a number of key elements, including the division of the search area, the representation of the data, the number of stages, the communication overhead, etc. Here, they are briefly covered:

Search Space Division: The partitioning of the search space among different nodes mimics how computations work as a whole. Subtasks are created within the dataset or problem. The input database is used to create a number of projected databases, which are then divided among the nodes using the division method. In order for each projected database to generate local itemsets independent of the actions of other nodes, it must contain fundamental information. The ultimate result is created by combining the local results of each subtask to produce the final output.

Database Layout: For parallel mining to succeed, data representation is crucial. Data representations come in two flavors: horizontal layout and vertical layout. Data is often stored in rows in a horizontal structure. Data is stored in columns as key-value pairs in a vertical style.

The process's speed, scalability, and overall design are all significantly impacted by the arrangement. The number of database scans in some algorithms is also determined by the layout.

Communication Cost: In parallel programming, the communication overhead is crucial because it can impact the entire processing speed. In the event that large files are transferred often, the network may get congested. A good paradigm must lower the cost of transmission.

Load Balancing: In distributed processing, load balancing is crucial since it has an impact on the overall effectiveness of the calculation. The strategic importance of fair distribution in balanced processing,however, the subtasks might have varying computational complexity, which might result in varying time complexity. While some nodes might be idle, others might be overloaded. In these situations, dynamic load balancing strategies are helpful in order to make the best use of the available resources.

6. Live streaming data with parallel mining

Several studies have already been done on extracting common patterns over streams. In order to keep a set of itemsets updated dynamically for future mining, Chi et al. [11] devised a small data structure. On the basis of a pane-wised sliding window, Tanbeer et al[14] .'s algorithm for mining frequent patterns was presented. However, the data is expanding at an unprecedented rate as a result of the growth of the internet and of mobile devices. Due to their own limitations, the conventional single-machine frequent pattern mining algorithms were unable to adapt to the expansion of vast amounts of data. It is becoming standard practice in this sector to create and implement distributed, parallel frequent pattern mining algorithms using the already-existing parallel, distributed framework [40] [42] [41] [43] [44].

There has recently been a lot of development in the field of stream data mining [45], which involves adapting common statistical and data analysis techniques to data streams or related models (e.g. [46]). Likewise, a number of techniques for online data mining have been presented (e.g. [52], [47], [48], [6], [17], [47]). A number of papers, including [20], [44], and [56], have specifically addressed the clustering issue in relation to data streams. Unlike our challenge, which requires clustering the streams themselves rather than individual data items, the problem

in these studies is to group the components of a single data stream. Our best understanding indicates that the literature has not yet addressed the issue in this form.
Time series data mining in general and time series clustering, in particular, have received a lot of attention [53]. Although time series data mining and stream data mining are undoubtedly related, there are still significant differences between these two disciplines. When compared to data streams, where the emphasis is on dynamic adaptation and online data mining, time series are still particularly static objects that can only be studied offline.

A parallel, apriori approach using the MapReduce Framework was proposed by Lin et al. [42]. An extensive dataset may be mined with this multi-scan approach. The parallel algorithm was created by Li et al. [43] in a similar manner to Lin et al. [42], with a few implementation-related differences. The MRApriori algorithm, which can locate common item sets in datasets with a horizontal or vertical format, was proposed by Hammoud et al. [44]. On distributed machines referred to as Pfp, Li et al[6] .'s proposal to parallelize the FPGrowth algorithm [45] was made. The machine learning (ML) library in Spark has this algorithm implemented. Tovarying degrees, the algorithms outlined above struggle to adapt to enormous data streams.

One of the key causes is that these parallel algorithms must do numerous data scans when mining the common itemsets. This intrinsic difficulty limits the benefits of mining huge data streams. As we have already stated, it is preferable to mine data streams incrementally. For instance, it is preferable to start new iteration procedures using some data from previous iterations. Once the dataset has been scanned several times, the non-frequent item sets and associated counting information are discarded by traditional frequent pattern mining techniques on static data [49].

However, because they must be accurate and scalable, concurrent data structures are difficult to design and implement [36]. Such data structures' safety and liveness qualities [37] serve as evidence for their correctness. While the liveness property describes the progress promises by demonstrating that "something positive will eventually happen," the safety property describes the consistency guarantees by demonstrating that "something negative will not happen." For the safety property, various formalizations, including linearizability [38] and sequential consistency [39], are offered in the literature.

Linearizability maintains the operations' real-time occurrence and mandates that each operation take effect instantly at a specific point between its invocation and response. If there is an ordered history of invocation and response events (i.e., an execution that can be linearized), then all processes must have the ability to organize the latter in real-time. The actual sequence of occurrences at various stages, however, may not always be important. Sequential consistency is employed as the safety criteria in these circumstances. Sequential consistency keeps the program order of operations produced by the same process, as opposed to the actual sequence of events.

When mining frequent itemsets in a data stream, this method won't work since the counts of the itemsets fluctuate constantly as fresh stream data is continuously added. Therefore, as new stream data arises, some objects that are currently uncommon may become so once more. Therefore, these traditional methods are unable to gather and preserve the smallest amount of data that is necessary to mine frequently occurring itemsets over time. an algorithm for mining frequent patterns in data streams with one pass that is parallel. The experiment demonstrates that it can increase speed over big datasets. The balance problem is another difficulty with parallel and distributed algorithms for mining common patterns.

Since every execution that is linearizable also offers sequential consistency, but the opposite is not always true, consecutive consistency is a weaker criterion than linearizability. Similar to how the wait-freedom and lock-freedom properties are stated, the liveness property is also specified through various formalizations [50, 51]. The biggest progress guarantee when it comes to personal success is wait-freedom. It ensures that irrespective of delay or failures in other processes, each process has a limit to the amount of steps required for its operation to be completed. Lock-freedom ensures system progress by guaranteeing a process will finish operating after a certain number of steps. Many efforts have been undertaken in recent years to create data structures that are more practical and efficient. [52].

Without using an appropriate grouping method, Pramudiono et al. [46] showed that the workload distributed across different nodes varies significantly. They proposed a technique to divide the large load into multiple minor loads with a predetermined threshold based on this discovery.

Using a granularity control mechanism and tree remerging, Pramudiono et al. [46] devised a method to address the imbalance problem. The framework will move a heavy workload from one node to another if it exceeds the predetermined threshold for that node. The burden in a stream of data, however, is constantly changing over time due to the continuous, high-speed nature and unbounded nature of the data stream, making a fixed threshold setting inappropriate in this situation.

The most recent hardware innovations influence all computational devices, including embedded resource-constrained ones, and are not just relevant to high-end servers (such as Intel Xeon Phi [53]). (e.g. Odroid XU4 [54]). As an illustration, multicore architecture is now being employed by embedded devices used for edge computing to facilitate parallel programming [54]. This serves as the impetus for the creation and application of high-level parallel and heterogeneous programming models that can be applied to any platform in the IoT architecture [55,56].

After analyzing the previously described PfP imbalance problem, Zhou et al. [47] presented a balanced partitioning approach based on F-lists. Two presumptions form the foundation of the partitioning strategy. In the beginning, the load of FP-Growth on each item's conditional pattern base is estimated using the number of recursive iterations that occurred during the execution of FPGrowth on those bases. Second, each item in the F-List is placed at an estimated distance equal to the length of the longest common pattern path in the conditional pattern base. When used in the context of a data stream, the assumption is unsuitable. The order of the items in the F-list will typically change significantly when the stream arrives continually (second assumption), therefore it is difficult to determine which pattern path in the conditional pattern base is the longest.

7. Limitation of the review

The creation of tools and improvements to computing infrastructures increase parallelism while maintaining determinism with an emphasis on two problem domains:

- continuous processing and analysis, and
- elasticity in stream processing.

In general, studies show a trade-off between performance and accuracy for continuous analysis. Data is generally collected in batches before clustering is done for particular applications where the results need to be accurate.

The difficulties with stream analysis of huge data revolve around:

- Evaluating the value of mined patterns
- Creating a universal theory of big data mining
- Scaling up to accommodate the expanding demands of enormous data sets
- Developing a powerful platform for large data mining
- Creating effective big data mining algorithms and models
- Upholding security, trust, and data integrity
- Issues with data privacy.

8. Conclusion and further work

Stream processing is becoming more and more necessary. Not only does an organization or enterprise needs to be able to process a vast volume of data fast to adapt to changing situations in real-time. Before conducting analysis, data streaming can be used for data validation and quality enhancement. Before uploading via a communications network to other tiers or saving in the database, this preprocessing strategy can enhance the analysis's quality and reduce the size of the data.

Load balancing is essential for parallel computing, and the data load must be divided equally among the processors to avoid making certain processors work harder than others. In this case, it is not necessary to review and process the records in these nodes. The data load must be distributed equally among the processors in order to achieve load balancing in parallel computing; otherwise, some processors would work harder than others. The records in these nodes don't need to be examined and processed in this situation. When certain nodes reach a dead end and some processors are processing more than the rest, the algorithm experiences an unbalanced difficulty that lengthens its runtime.

Data scientists are now more interested in using cutting-edge, powerful deep learning (DL) algorithms. DL is effective in handling big data analytics. It can be extensively used to handle certain common Big Data Analytics issues, including data labeling, semantic indexing, sifting through enormous volumes of data to find complicated and nonlinear patterns, quick and

effective information retrieval, and the simplification of discriminatory jobs. It would therefore be a progressive approach to our work in the future.

REFERENCES

[1] "How Much Data Is Created Every Day," 28 October 2021. [Online]. Available: https://seedscientific.com/how-much-data-is-created-every-day/. [Accessed 18 12 2021].

[2] M. Garofalakis, J. Gehrke and R. Rastogi, in Data Stream Management: Processing High-Speed Data Streams, Springer, 2016.

[3] A. Bifet, R. Gavalda, G. Holmes and B. Pfahringer, Machine Learning for Data Streams:with Practical Examples in MOA, MIT Press, 2018.

[4] P. K. SRIMANI and M. M. PATIL, "Mining data streams with concept drift in massive online analysis frame work," WSEAS Trans. Comput, vol. 15, pp. 133-142, 2016.

[5] Kamran Soomro, Zaheer Khan, and KhawarHasham. Towards provisioning of real-time smart city services using clouds. In Proceedings of the 9th International Conference on Utility and Cloud Computing, pages 191–195. ACM, 2016.

[6] Christos Vlassopoulos, IoannisKontopoulos, MichailApostolou, Alexander Artikis, and DimitriosVogiatzis. Dynamic graph management for streaming social media analytics. In Proceedings of the 10th ACM International Conference on Distributed and Event-based Systems, pages 382–385. ACM, 2016.

[7] E. Frank, . M. A. Hall and I. H, "Weka 3: Machine Learning Software in Java," University of Waikato, 2016. [Online]. Available: https://www.cs.waikato.ac.nz/ml/weka/. [Accessed 2022].

[8] A. Hadoop®, "What Is Apache Hadoop?," hadoop.apache.org, 18 10 2018. [Online]. Available: http://hadoop.apache.org/. [Accessed 2 12 2018].

[9] "Apache Spark™ - Lightning-Fast Cluster Computing," The Apache Software Foundation, 23 2017. [Online]. Available: https://spark.apache.org/. [Accessed 2 12 2018].

[10] "Apache Storm," Apache Software Foundation, 3 6 2017. [Online]. Available:http://storm.apache.org/. [Accessed 2 12 2018].

[11] Bange, D., Grosser, T. and Janoschek, N. (2013). "BIG DATA SURVEY EUROPE - Usage, technology and budgets in European best-practice companies". [online] Wuerzburg, Germany: BARC Institute. Available at:

http://www.pmone.com/fileadmin/user_upload/doc/study/
BARC_BIG_DATA_SURVEY_EN_final.pdf [Accessed 27 Jan. 2016]

[12] Toshniwal, A., Taneja, S., Shukla, A., Ramasamy, K., Patel, J. M., Kulkarni, S. &Bhagat, N. (2014). "Storm@ twitter". In Proceedings of the 2014 ACM SIGMOD international conference on Management of data, pp. 147-156. ACM.

[13] Companies Using Apache Storm: https://storm.apache.org/documentation/Powered-By.html

[14] Zaharia, M., Chowdhury, M., Franklin, M. J., Shenker, S., &Stoica, I. (2010). "Spark: cluster computing with working sets". In Proceedings of the 2nd USENIX conference on Hot topics in cloud computing, vol. 10, pp. 10.

[15] YARN: https://hadoop.apache.org/docs/current/hadoop-yarn/hadoop-yarn-site/ YARN. html

[16] Zaharia, M., Das, T., Li, H., Shenker, S., &Stoica, I. (2012). Discretized streams: an efficient and fault-tolerant model for stream processing on large clusters. In Proceedings of the 4th USENIX conference on Hot Topics in Cloud Computing, pp. 10-10. USENIX Association.

[17] A. Bifet, R. Gavalda, G. Holmes and B. Pfahringer, Machine Learning for Data Streams:with Practical Examples in MOA, MIT Press, 2018.

[18] Gama, J., `Data stream mining: the bounded rationality´, Informatica Vol. 37, No 1, 2013, pp. 21-25. - Table

[19] "Apache Flink: Introduction to Apache Flink®," The Apache Software Foundation, 7 2 2017. [Online]. Available: https://flink.apache.org/introduction.html. [Accessed 2 12 2018].

[20] Krawcyz, B., Minku, L.L., Gama, J., Stefanowski, J., Wozniak, M. 2017. `Ensemble learning for data stream analysis: a survey´, Information Fusion, Vol. 37, 2017, pp. 132-156. - Table

[21] Nguyen, H.L., Woon, Y.K., Ng, W.K., `A survey on data stream clustering and classification´, Knowledge and Information Systems, Vol., 45, No 3, 2015, pp. 535-569. - Table

[22] https://flink.apache.org/downloads.html#apache-flink-ml-200

[23] Tyler Akidau and Alex Balikov and Kaya Bekiroglu and SlavaChernyak and Josh Haberman and Reuven Lax and Sam McVeety and Daniel Mills and Paul Nordstrom and Sam Whittle , “MillWheel: Fault-Tolerant Stream Processing at Internet Scale”, Very Large Data Bases,pp.734—746, 2013.

[24] Ren X, Khrouf H, Kazi-Aoul Z, ChabChoub Y, Cure O. On measuring performances of C-SPARQL and CQELS. CoRR, abs/1611.08269. 2016.

[25] Inoubli W, Aridhi S, Mezni H, Maddouri M, Nguifo E. A comparative study on streaming frameworks for big data. In: 44th international conference on very large databases: workshop LADaS—Latin American Data Science, Aug 2018, Rio de Janeiro, Brazil. 2018. p. 1–8.

[26] Liao X, Gao Z, Ji W, Wang Y. An enforcement of real-time scheduling in Spark Streaming. 6th international green and sustainable computing conference, IEEE. 2016. https://doi.org/10.1109/igcc.2015.7393730. p. 1–6.

[27] Vanathi R, and Khadir ASA. A robust architectural framework for big data stream computing in personal healthcare real-time analytics. World Congress on Computing and Communication Technologies. 2017. p. 97–104. https://doi. org/10.1109/wccct.2016.32.

[28] Gokalp MO, Kocyigit A, Eren PE. A visual programming framework for distributed Internet of Things centric complex event processing. Comput Elect Eng. 2018;74:581–604. 96. Maio CD, Fenza G, Loia E, Orciuoli F. Distributed online temporal fuzzy concept analysis for stream processing in smart cities. J Parallel DistribComput. 2017;110:31–41.

[29] Fernandez-Rodrigues JY, Alvarez-Garcia JA, Fisteus JA, Luaces MR, Magana VC. Benchmarking real-time vehicle data streaming models for a smart city. Inform Syst. 2017;72:62–76.

[30] J. Chen, K. Li, Z.Tang, K. Bilal, S. Yu, C. Weng, and K. Li. "A parallel random forest algorithm for big data in a spark cloud computing environment." IEEE Transactions on Parallel and Distributed Systems, vol. 28, no. 4, pp. 919-933, 2017.

[31] Ma K, Yang B. Stream-based live entity resolution approach with adaptive duplicate count strategy. Int J Web Grid Serv. 2017;13(3):351–73.

[32] Troiano L, Vaccaro A, Vitelli MC. On-line smart grids optimization by case-based reasoning on big data. In: 2016 IEEE workshop on environmental, energy, and structural monitoring systems (EESMS), Bari, Italy, 13–14 Jun 2016.

[33] Murphy BM, O'Driscoll C, Boylan GB, Lightbody G, Marnane WP. Stream computing for biomedical signal processing: A QRS complex detection case study. In: Confproc IEEE eng med biolsoc. 2015. https://doi.org/10.1109/ embc.2015.7319741. p. 5928–31.

[34] Apache Storm.: https://storm.apache.org/

[35] Samza: https://samza.apache.org/

[36] F. Zhu, Z. Li, S. Chen and G. Xiong. "Parallel transportation management and control system and its applications in building smart cities." IEEE Transactions on Intelligent Transportation Systems, vol.17, no. 6, pp. 1576-1585, 2016.

[37] Sakr S. An introduction to Infosphere streams: A platform for analysing big data in motion. IBM. 2013. https:// www.ibm.com/developerworks/library/bd-streamsintro/index.html. Accessed 7 Oct 2018.

[38] Xhafa F, Naranjo V, Caballé S. Processing and analytics of big data stream with Yahoo!S4. In: 2015 IEEE 29th international conference on advanced information networking and applications, Gwangiu, South Korea, 24–27 March 2015. 2015. https://doi.org/10.1109/aina.2015.194.

[39] Cortes R, Bonnaire X, Marin O, Sens P. Stream processing of healthcare sensor data: studying user traces to identify challenges from a big data perspective. The 4th international workshop on body area sensor networks (BASNet-2015). ProcediaComput Sci. 2015;52:1004–9.

[40] S. Moens, E. Aksehirli, and B. Goethals, "Frequent itemset mining for big data," in Big Data, 2013 IEEE International Conference on. IEEE, 2013, pp. 111–118.

[41] W. Fan and A. Bifet, "Mining big data: current status, and forecast to the future," ACM sIGKDD Explorations Newsletter, vol. 14, no. 2, pp. 1–5, 2013.

[42] M.-Y. Lin, P.-Y. Lee, and S.-C. Hsueh, "Apriori-based frequent itemset mining algorithms on mapreduce," in Proceedings of the 6th international conference on ubiquitous information management and communication. ACM, 2012, p. 76.

[43] N. Li, L. Zeng, Q. He, and Z. Shi, "Parallel implementation of apriori algorithm based on mapreduce," in Software Engineering, Artificial Intelligence, Networking and Parallel

&Distributed Computing (SNPD), 2012 13th ACIS International Conference on. IEEE, 2012, pp. 236–241.

[44] S. Hammoud, “Mapreduce network enabled algorithms for classification based on association rules,” Ph.D. dissertation, Brunel University School of Engineering and Design PhD Theses, 2011.

[45] J. Han, J. Pei, and Y. Yin, “Mining frequent patterns without candidate generation,” in ACM Sigmod Record, vol. 29, no. 2. ACM, 2000, pp. 1–12.

[46] I. Pramudiono and M. Kitsuregawa, “Tree structure based parallel frequent pattern mining on pc cluster,” in International Conference on Database and Expert Systems Applications. Springer, 2003, pp. 537–547.

[47] L. Zhou, Z. Zhong, J. Chang, J. Li, J. Z. Huang, and S. Feng, “Balanced parallel fp-growth with mapreduce,” in Information Computing and Telecommunications (YC-ICT), 2010 IEEE Youth Conference on. IEEE, 2010, pp. 243–246.

[48] M. Herlihy and N. Shavit, The art of multiprocessor programming. Morgan Kaufmann, 2011.

[49] D. Cederman, A. Gidenstam, P. Ha, H. Sundell, M. Papatriantafilou, and P. Tsigas, “Lock-free concurrent data structures,” Programming Multicore and Many-core Computing Systems, vol. 86, p. 59, 2017.

[50] D. Cederman, B. Chatterjee, N. Nguyen, Y. Nikolakopoulos, M. Papatriantafilou, and P. Tsigas, “A study of the behavior of synchronization methods in commonly used languages and systems,” in 2013 IEEE 27th International Symposium on Parallel and Distributed Processing. IEEE, 2013, pp. 1309–1320.

[51] Y. Nikolakopoulos, A. Gidenstam, M. Papatriantafilou, and P. Tsigas, “A consistency framework for iteration operations in concurrent data structures,” in 2015 IEEE International Parallel and Distributed Processing Symposium. IEEE, 2015, pp. 239–248.

[52] V. Gulisano, Y. Nikolakopoulos, D. Cederman, M. Papatriantafilou, and P. Tsigas, “Efficient data streaming multiway aggregation through concurrent algorithmic designs and new abstract data types,” ACM Transactions on Parallel Computing (TOPC), vol. 4, no. 2, p. 11, 2017.

[53] Intel. (2019) Xeon-phi. https://www.intel.com/.

[54] Hardkernel. (2019) Odroid-xu4. https://www.hardkernel.com.

[55] A. Ernstsson, L. Li, and C. Kessler, “Skepu 2: Flexible and type-safe skeleton programming for heterogeneous parallel systems,” International Journal of Parallel Programming, vol. 46, no. 1, pp. 62–80, 2018.

[56] Y. Nikolakopoulos, M. Papatriantafilou, P. Brauer, M. Lundqvist, V. Gulisano, and P. Tsigas, “Highly concurrent stream synchronization in many-core embedded systems,” in Proceedings of the Third ACM International Workshop on Many-core Embedded Systems. ACM, 2016, pp. 2–9.

[57] I. Walulya, Y. Nikolakopoulos, M. Papatriantafilou, and P. Tsigas, “Concurrent data structures in architectures with limited shared memory support,” in European Conference on Parallel Processing. Springer, 2014, pp. 189–200.