Jordanian Journal of Computers and Information Technology (JJCIT), Vol. 08, No. 03, September 2022.

A NOVEL TRUE-REAL-TIME SPATIOTEMPORAL DATA STREAM PROCESSING FRAMEWORK

Ature Angbera[1] and Huah Yong Chan[2]

(Received: 9-Mar.-2022, Revised: 2-May-2022, Accepted: 23-May-2022)

ABSTRACT

The ability to interpret spatiotemporal data streams in real time is critical for a range of systems. However, processing vast amounts of spatiotemporal data out of several sources, such as online traffic, social platforms, sensor networks and other sources, is a considerable challenge. The major goal of this study is to create a framework for processing and analyzing spatiotemporal data from multiple sources with irregular shapes, so that researchers can focus on data analysis instead of worrying about the data sources' structure. We introduced a novel spatiotemporal data paradigm for true-real-time stream processing, which enables high-speed and low-latency real-time data processing, with these considerations in mind. A comparison of two state-of-the-art real-time process architectures was offered, as well as a full review of the various open-source technologies for real-time data stream processing and their system topologies were also presented. Hence, this study proposed a brand-new framework that integrates Apache Kafka for spatiotemporal data ingestion, Apache Flink for true-real-time processing of spatiotemporal stream data, as well as machine learning for real-time predictions and Apache Cassandra at the storage layer for distributed storage in real time. The proposed framework was compared with others from the literature using the following features: Scalability (Sc), prediction tools (PT), data analytics (DA), multiple event types (MET), data storage (DS), Real-time (Rt) and performance evaluation (PE) stream processing (SP) and our proposed framework provided the ability to handle all of these tasks.

KEYWORDS

Spatiotemporal big data, Real-time processing, Stream processing, Apache Kafka, Apache Flink, Apache Cassandra, Apache Spark.

1. INTRODUCTION

One of the new elements for Internet-based applications is spatial-temporal data. Data utilized for real-time analytics has become a part of production data in new internet application trends [1]. Spatiotemporal data is in enormous quantities through a variety of activities, such as clicks on a social-media platform orders, sales, shipment data in retail and so on. As the internet-connected world grows exponentially, a vast volume of data is generated in a continuous stream from a variety of sources. Five billion people utilize various varieties of mobile devices, according to [2]. Again, according to an IBM big-data study, there will be around 35 zettabytes of data generated yearly from 2020 [3] and data growth will be 50 times faster than it is presently [4]. Every day, 2.5 quintillion bytes of information are created [3]. The amount of data that can be generated is limitless. This phenomenon is referred to as "big data" [1]. In conjunction with this concept, the 3V's of big data (Volume, Velocity and Variety) have been coined [5]. The first V stands for the massive amounts of data produced by these technologies. The second V represents the rate at which the sources generate these data, while the third V represents the data's heterogeneity. There is usually a large amount of spatial-temporal data [6] that is being generated. The 3.8 billion individuals and 8.06 billion internet-connected devices are responsible for the vast amounts of data produced [7]. According to [8], one zettabyte of data was generated in 2010 and seven zettabytes in 2014. As a result of the rapid emergence of massive spatiotemporal datasets, volume, variety and velocity are all concerns that must be addressed. The phrase "spatiotemporal big-data volume" refers to a large number of data that necessitates a great amount of processing and storage [9]. The volume of data is increasing faster than computational processing devices can keep up [10]. Spatiotemporal data is a continuous stream of data in terms of velocity. As a result, the concerned stakeholders have to spend a lot of money on processing [11]. Real-time data stream processing is critical for a variety of applications, but processing a massive volume of data coming from several sources, like online traffic, sensor networks and many other sources, is a significant issue [3]. The most serious problem has been that the spatiotemporal big-data

1. A. Angbera is with School of Computer Sciences, Universiti Sains Malaysia, Pulau Pinang 11800, Malaysia. And with Department of Computer Science Joseph Sarwuan Tarka University, Makurdi, Nigeria. Email: angberaature@student.usm.my
2. H. Y. Chan is with School of Computer Sciences, Universiti Sains Malaysia, Pulau Pinang 11800, Malaysia. Email: hychan@usm.my

system is based around "Hadoop"; namely "MapReduce" [12]-[13]. This framework has a high level of scalability and fault tolerance. A vast volume of data in batches can be managed by this system and it provides observation blow understanding of past data, but it can only analyze a constrained collection of data. Although MapReduce [14]–[19] isn't designed for processing real-time stream applications, its critical function is to process data as soon as it arrives to get a quick response and make effective decisions. As a result, for effective and speedy analysis having very low latency and high throughput, a real-time stream processing spatiotemporal data novel framework is required.

From the literature, it is observed that there is a vast growth of spatiotemporal datasets which are coming from different sources, resulting in different structures of spatiotemporal datasets. This leads to a serious problem known as heterogeneity of spatiotemporal data [6]. Processing of these spatiotemporal datasets is challenging, as they do not have a common data structure presentation for modeling, resulting in inaccurate results during data analytics. Also, it was observed that spatiotemporal big-data systems are based around “Hadoop”, which is not designed to process data in real time; hence, in our study, we proposed a true-real-time stream processing framework for spatiotemporal data, that takes care of the heterogeneous data issues and processes big spatiotemporal data in real time.

This paper gives an overview of certain essential concepts in spatiotemporal unbounded data, stream processing, big-data storage and other related fields. After that, we present some tools and systems that enable real-time data processing. A full comparison of various queueing message systems, stream processing platforms and data storage platforms is also made available. In addition, we offer a novel framework based on the previous comparison. Finally, we compare and contrast our suggested novel framework with others from the literature. The rest of the paper is structured as follows: Section 2 highlights spatiotemporal big data. Data-processing technologies are presented in Section 3. Section 4 presents the distributed queuing management technologies with a comparison among them. Section 5 presents the technologies for big-data storage. The state-of-the-art real-time processing architectures are highlighted and compared in Section 6. Section 7 presents the proposed framework and Section 8 provides the conclusions.

2. SPATIOTEMPORAL BIG DATA

In terms of spatial data storage, geographic analysis and spatiotemporal visualization, a lot of big-data platforms fared poorly [20]. In industry, research and a range of other fields, the term "big data" is now commonly used. "Spatiotemporal big data" describes the creation of huge datasets, which are unable to be analyzed and managed due to the vast amount and complexity of data collected at a certain given time. As information technology has progressed, the demand for processing, understanding and displaying spatiotemporal large data has increased significantly. A vast volume of geographical and spatiotemporal data, as well as its application sectors, necessitate a distributed and highly scalable data-processing system [21]. This demand is being met by researchers from both academia and industry. The MapReduce framework, Hadoop [22], NoSQL databases [23] and Spark [24] are all used in modern large-scale geographic data processing systems. The majority of these systems improved previous systems by adding spatial or spatiotemporal support as a layer on top of them or by extending the core of them. A high number of these systems were built from the ground up or run on platforms other than Hadoop, NoSQL and Spark. A true-real-time stream processing spatiotemporal data framework having these properties; namely, scalability (Sc), prediction tools (PT), data analytics (DA), multiple event types (MET), data storage (DS), Real-time (Rt) and performance evaluation (PE) stream processing (SP), is rare in the literature.

- Stream Processing (SP): To process data that just comes in, the system should use stream processing. Stream-processing technologies enable us to transform and analyze data as it is being received. As a result, stream processing is critical for delivering real-time functionality [25].
- Data Storage (DS): To save essential data, the system needs to include a data-storage layer. This data can be used in other operations as well as for historical analyses. Furthermore, storage systems should be able to store a vast amount of data as a result of the size of spatiotemporal data [26].

- Performance Evaluation (PE): To demonstrate the proposed system's correctness and effectiveness, an assessment should be available. In many cases, performance is critical; as a result, it should've been evaluated to present enterprises with adequate knowledge to measure the success of the suggested solution and its suitability for the current circumstances [25].
- Multiple Event Types (MET): The technology should support a wide range of kinds of events as input sources, including various forms, such as XML, Java Map and JSON and also as unstructured forms, such as text, CSV and actual data. This characteristic will enable the framework to manage and incorporate a diverse set of spatiotemporal data sources, allowing it to be used in real-world settings [25].
- Scalability (Sc): Modern large-data stream processing engines stress scalability as a fundamental quality attribute [27]. The system must be scalable; this is to aid a high volume of data control. In most cases, more data sources can be incorporated into the same infrastructure and this should be able to handle the influx of new spatiotemporal data [25].
- Prediction Technologies (PT): The technology could also provide developers with prediction tools to help them enhance the level of service and the outcomes [25].
- Data Analytics (DA): This spatiotemporal (heterogeneous) data must be analyzed by at least one mechanism in the system. The capacity to evaluate the data we collect to uncover and report circumstances of interest to interested agents is one of the most significant benefits of the internet of things [26].
- Real-time Processing (RP): Spatiotemporal data should be processed in real time or close to real time. This kind of system will be able to respond to situations of interest as early as possible and as effectively as possible if we can conduct all of the supplied functionality immediately [25]-[26].

3. DATA PROCESSING TECHNOLOGIES

3.1 Data Stream Processing

Real-time data analytics with low latency and high throughput needs became increasingly important in many sectors, such as healthcare, transportation and smart homes [28]. In the industry, stream processing is getting a lot of popularity as a new programming paradigm for implementing real-time data-driven applications [29]. “A stream is an infinite series of tuples in a distributed data stream processing system (DSPS). A data source reads data from an external source (or sources) and feeds it into the system as streams of data. A processing unit (PU) takes tuples from data sources or other PUs and processes them with user-supplied code. It can then transfer the data to other PUs for further processing [30]. To express parallelism, a DSPS typically uses two levels of abstraction (logical and physical). An application is typically depicted as a directed graph in the logical layer, with each vertex corresponding to a data source or a PU and direct edges indicating how data tuples are transmitted between data sources/PUs. Each data source or PU can run as many parallel jobs as possible on a cluster of machines and each task is an instance of that data source or PU. A DSPS's physical layer typically consists of a group of virtual or physical machines that process data received and a master that acts as the cluster's central control unit, distributing user code, scheduling jobs and monitoring them for problems. An application graph is run on numerous worker processes on multiple (physical or virtual) machines at runtime. In most cases, each machine is set up with many slots. The number of slots specifies how many worker processes can execute on this machine and can be pre-configured by the cluster operator based on hardware constraints (such as the number of CPU cores)”. Each worker process has its slot, which is used to process data tuples using user code utilizing one or more threads. Normally, at runtime, a job is assigned to a thread (even if it does not have to be this way). A scheduling mechanism in a DSPS outlines how threads are assigned to processes and machines. A default scheduler is included with many DSPS; however, it can be modified with a custom scheduler. The default scheduler often employs a straightforward scheduling approach that distributes threads to pre-configured processes, which are subsequently assigned to machines in a round-robin fashion. This technique results in a nearly even workload distribution throughout the cluster's available machines. In addition, a DSPS usually provides multiple grouping options, which specify how tuples are distributed among tasks [30].

3.2 Stream Processing Platforms

In this part, we explore and present the differences between data stream processing tools, such as Apache Spark, Apache Hadoop, Apache Storm and Apache Flink. To organize and analyze data, classic relational database management systems, as well as many current batch processing tools, like Hadoop and Spark, have been deployed. Although these technologies have progressed and are beneficial for several products, they are not the greatest choice for creating real-time applications [31]. As a result, emerging innovations, like Apache Storm, Apache Flink and others, have been developed to manage vast quantities of data streams, process them and analyze them as they move to accomplish the demands of real-time applications. These technologies strive to capture the importance of time in real-time analytics, streaming analytics and sophisticated-event processing. We are inspired to provide a truly real-time stream processing framework for spatiotemporal data because of the necessity of such emerging technologies. We will show that earlier techniques, such as MapReduce, do not provide real-time processing despite their capacity to process a vast volume of data, not minding the rate at which the data comes in.

Apache Spark: Spark is a unified large-data analytics engine with built-in streaming, SQL, machine learning and graph-processing modules. It was created at the University of California in 2009, released as an open source in 2010 and given to the Apache Software Foundation in 2013, which has been in charge of the project since then. Spark is the successor to Hadoop, which was the original big-data analytics platform and was used for batch processing [32]. The MapReduce paradigm typically employs a linear data flow to take data out of the disc, map a function across the data, reduce the results to that map and ultimately save this reduced result on the disc-inspired Spark. Furthermore, Spark's "Resilient Distributed Dataset" (RDD) enables multiple readings of datasets as well as interactive data analysis [33]. The Spark adds in-memory processing, which allows for up to 100 times quicker processing, albeit it has the drawback of requiring smaller datasets than Hadoop due to resource constraints. Spark's architecture is made up of Spark Core, which is the project's foundation and the modules or frameworks listed above, which are built on top of it: MLlib for machine learning, Spark Streaming, Spark SQL and GraphX for graph processing. Through an API based on the RDD abstraction, Spark Core provides basic I/O functionality as well as distributed task dispatching and scheduling. This construction is depicted in detail in Figure 1.

Apache Hadoop: It's a platform with open access for data processing that makes use of commodity technology to store and analyze enormous volumes of data. The Hadoop ecosystem is seen in Figure 2 along with the framework's major components. The "Hadoop Distributed File System" (HDFS) and the "MapReduce programming" style are the two most significant components of the Hadoop architecture. The data is stored in HDFS and processed in a distributed way using MapReduce. Despite its many benefits, Hadoop lacks storage and network encryption, has limited flexibility, is unsuitable for tiny-data collections and has a large I/O overhead. Hadoop, particularly the Map-Reduce framework, which is never the better technology for processing the most recent set of data, is constrained to batch processing. This is one of its major disadvantages [3].

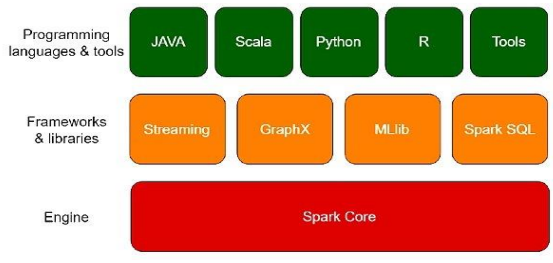

Figure 1. Apache Spark architecture.

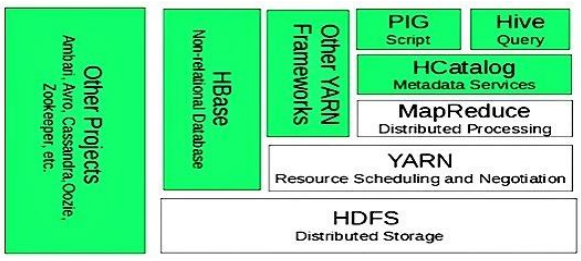

Figure 2. Hadoop ecosystem [3].

Apache Storm: It's an open-source distributed framework that makes it easier to create fault-tolerant programs that run in parallel on computing clusters [34]. The Storm was developed by BackType, a business that was acquired by Twitter in 2011. It is an Eclipse Public License-compliant open-source project. In Storm, a topology is a computing network (Figure 3) that defines how data (such as tuples) travels between processing units [35]. A topology can continue to run indefinitely or until it is interrupted by a user. Similar to earlier application designs, a topology gathers information and

separates it into portions that are handled by assignments to cluster nodes. Data that nodes share is tuples, which are sorted collections of values. The Storm is built on a master-slave paradigm, with a master node running the Nimbus daemon and keeping a membership list to ensure data-processing reliability. According to Nimbus, it connects to Apache Zookeeper [35].

A Storm cluster is comprised of 3 nodes, as illustrated in Figure 4: "Nimbus," (when the original Nimbus instance fails, the secondary Nimbus instance takes over [36]). That is the same as Hadoop's job tracker, "Supervisor," which is in charge of starting and halting the process and "Zookeeper," a common coordination server that governs the cluster of the Storm [3].

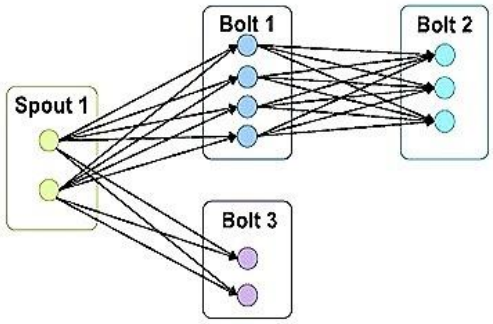

Figure 3. Topology of a storm.

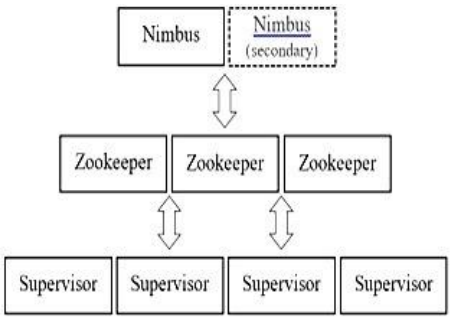

Figure 4. Storm Architecture [36].

Apache Flink: It's a platform for stream and batch data processing with open access, that arose from a fork of the "Stratosphere" project, which was founded in 2010 and developed by a team of researchers from Humboldt-Universität zu Berlin, Technical University Berlin and Hasso-Plattner-Institut Postdam with funding from the German Research Foundation. The project's goal was to develop a new big-data analytics platform to aid research in Berlin-area universities. It was elevated to a high-stage project at the "Apache Software Foundation" at the end of 2014 [32]. The master-slave model is the base design for Flink, which is made up of three primary components. Job Manager: It is the distributed execution's coordination node (master node) that manages the data flow between the slave nodes' task managers. The Task Manager is in charge of executing the operators that receive and produce streams, notifying the Job Manager of their status and exchanging data streams amongst the operators (task managers). Client: It converts computer code into a data-flow graph, which is then sent to the Job Manager to be executed. Flink is a native (true) stream-processing framework that can also handle batch processing, considering each batch as a stream of bounded data. Apache Flink combines stream processing with CEP (Complex-event Processing) [37] technology to provide real-time data analysis and response. Flink allows us to apply transformations to data streams and then analyze the results [38]. Figure 6 shows the ecosystem of flink.

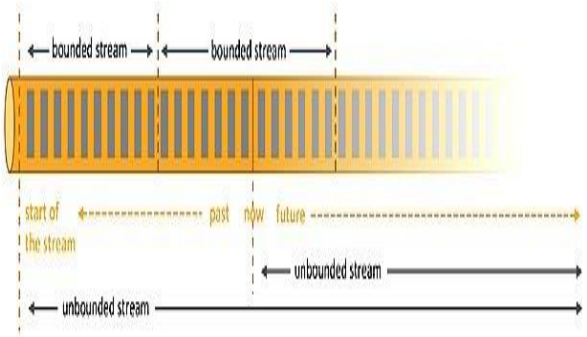

Figure 5. Structure of streams [32].

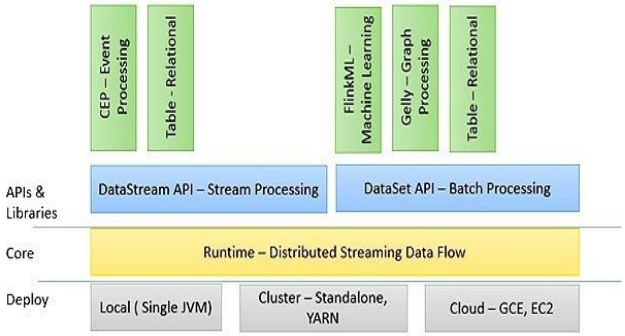

Figure 6. An ecosystem of Flink.

Streams that are unbounded and bounded are depicted in detail in Figure 5. Unbounded streams have a beginning, but no finish; and to achieve a complete result, the sequence in which the events are generated frequently matters. Bounded streams have a beginning and an end, but they may be sorted; thus, the order of events isn't important. Batch processing is the term for this method.

Table 1 lists the characteristics of the data-processing tools mentioned in this paper. For stream processing, Flink is a better technology for true real-time processing, Hadoop handles batch

processing and Spark can manage micro-batching, according to the comparison of several streaming data processing platforms offered in Table 1. To minimize the latency overhead that batching and micro-batching impose, Storm uses the spouts and bolts to execute one-at-a-time processing. Flink supports batch and true stream processing. It's highly optimized, with features such as light-weighted snapshots and it appears to be the data stream management system, the market leader. As we can see, the majority of the desired features (low latency, high throughput, guarantee of exactly-once execution and state management) are available. As a result, we adopt Flink as our computational framework for processing streaming data in our study.

Table 1. Data-processing technologies.

Features	**Apache Spark**	**Apache Flink**	**Apache Hadoop**	**Apache Storm**
Open access	Yes	Yes	Yes	Yes
Coordination tool	Zookeeper	Zookeeper	Zookeeper	Zookeeper
Language	Python, R, C#, Scala, Java	Scala, Python, SQL, Java	Scala, Python, Java	Any PL
In-memory processing	Yes	Yes	No	Yes
Data processing	Batch/Stream (micro-batch)	Batch/Stream (Native)	Batch	Streaming
Execution model	Micro-batch	Real-time (Ture streaming), micro-batch and batch.	Batch	Real-time (one at a time)
Fault tolerance	Yes	Yes	Yes	Yes
Achievable latency	Low latency	Lowest latency as compared with Spark and Storm	High	Very low latency
Data-processing guarantee	Exactly-once processing	Exactly-once processing	Exactly-once processing	At least once processing
Data storage	Yes	Yes	Yes	No
Optimization	Manual	Automatic	Manual	Manual
Operating system	Windows, macOS, Linux	Linux, macOS, Windows	UNIX, Windows	Windows, Linux, macOS
Throughput	High	Very high	Very low	Low

Flink has similar features to Spark, but it operates as a native stream engine, posing numerous obstacles to Spark in stream processing (e.g. in the case of latency and recovery). Flink also appears to be stronger than Storm [39].

4. DISTRIBUTED QUEUING MANAGEMENT TECHNOLOGIES

Data is transferred from one program to another using a messaging system. Applications can concentrate just on data rather than on how it is exchanged. Traditional messaging systems exist, but the majority of them are incapable of working with a huge volume of data in a real-time setting. Message queuing reliability is a key feature of distributed messaging systems. The P to P (point to point) pattern and the publish-subscribe pattern are the two types of message patterns. In a messaging system, the publish-subscribe, commonly known as pub-sub, is used [1]. Publish/subscribe messaging has been supported by distributed queue management solutions, like RabbitMQ, Amazon Kinesis, Kafka and Google Pub/Sub in recent years [31]. When it comes to transferring massive amounts of data around for real-time applications, these technologies have provided some beneficial new solutions. While distributed queue management systems may appear to be identical to traditional message queuing technologies, their architecture is vastly different and as a result, their performance and behavioral properties are vastly different. Traditional queuing schemes, for example, eliminate handled responses out from the queue and are unable to spread out when multiple consumers perform different activities at the same time. Distributed queuing systems, on the other hand, are well-suited for both online and offline content ingestion, because they can accommodate numerous clients and prevent data loss by distributing resilient discs across replicated clusters. The responses are committed to the dispersed queues as soon as feasible, ensuring message delivery for a set amount of time. Each distributed queue management solution splits its topics (i.e., where a producer publishes data

(messages) and a consumer retrieves it). The messages are absorbed by every consumer segment (partitions) of a specific subject, with just a single consumer from the same consumer segment consuming the same partition. The consumer group's function is quite beneficial for re-balancing when partitions and/or customers change [31]. Table 2 lists aspects to consider when selecting a distributed queuing system, involving messaging guarantees, disaster recovery, replication, federated queues (which disperse a single queue's load across nodes or clusters), supported languages and many others.

Table 2. Distributed message queuing technologies.

Features	**Kinesis**	**Ms. Azure Event Hub**	**Apache Kafka**	**RabbitMQ**	**Google pub/sub**
Supported language	Java, Python, .NET, C++, Go, PHP, Ruby, Node.js	Java, C++, Ruby, PHP, Node.js, Python, .NET	PHP, Ruby, Java, Python, .NET, Node.js, Go, C++	Go, C++/C, Java, Python, .NET, PHP, Ruby, Node.js	PHP, Ruby, Java, C++, Node.js, Python, .NET
Messaging guarantees	Yes / At least once	Yes / At least once	Yes / At least once	Yes / At least once	Yes / At least once
Configurable persistence period	from one to seven days (default is 24 hours)	24 hours as default (from one to seven days)	No maximum	N/A	Seven days (non-configurable) or only when it is recognized by all subscribers
Latency	200 ms to 5 seconds	There are no values cited.	Some set-ups are measured in ms. Benchmarking revealed a median delay of ~2 ms.	There are no values cited.	There are no values cited.
Recovery of disaster	Yes	Yes	Yes	Yes	Yes
Replication	Hidden (across three zones)	Configurable replicas	Configurable replicas	Configurable replicas	Hidden
Consumer groups	Yes	Yes	Yes	Yes	Yes
Guarantees ordering	Guaranteed within the confines of a partition	Guaranteed within the confines of a partition	Guaranteed within the confines of a partition	Guaranteed using AMQP channel	No order guarantees
Throughput	1 MB/s input, 2 MB/s output or 1000 records per second can all be supported by a single shard. 20,000 messages per second throughput	Throughput units have been scaled. Each one can handle 1 MB/s entrance, 2 MB/s egresses and 84 GB of storage. The standard tier allows for a total of 20 throughput units.	30,000 messages per second throughput	There are no figures for throughput that have been mentioned.	The standard is 100 MB/s in and 200 MB/s out; however, the maximum speed is stated to be infinite.

Apache Kafka is a real-time communication system that uses a distributed publish-subscribe model. Kafka can handle a large volume of data, allowing you to send messages at the end-point. We also choose Kafka over other popular messaging systems in this study for three reasons: To begin with, other similar message broker technologies, such as RabbitMQ, Amazon Kinesis, ActiveMQ and other enterprise messaging systems, are ephemeral, meaning that they keep data in memory or other light storage. Kafka, on the other hand, provides durability by persisting data on storage, which expands and broadens its application scenarios. Second, Kafka is a data-transit technology rather than a data-processing system. This distinguishes it from the competition in stream processing. The third reason is that Kafka is frequently used in conjunction with other systems for streaming-data processing.

For ingestion, Apache Kafka is currently state-of-the-art. To consume Kafka, two sets of actors are

required, as shown in Figure 7. **Producers** distribute messages on one or more Kafka topics. Data is sent to Kafka brokers by the producers. When a producer sends a message to a broker, it is considered published. Producers have the option of sending messages to a certain partition. **Consumers** are in charge of pulling data from Kafka brokers and sending it to processing nodes (e.g. Spark or Flink). Kafka brokers are managed and coordinated by a Zookeeper. When the latest broker is deployed to the Kafka system or when a broker in the Kafka system fails, the Zookeeper service is used to notify producers and consumers.

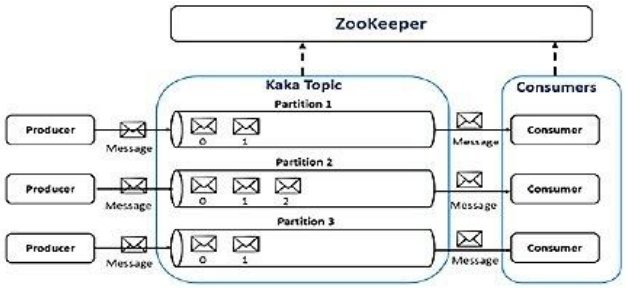

Figure 7. Apache Kafka framework.

5. TECHNOLOGIES FOR BIG DATA STORAGE

The difficulty of huge spatiotemporal data quantities that are growing at an exponential rate has lately been addressed by upgrading big-data analysis tools. The big-data analysis solutions often handle several issues, by giving the distributed environment the chance to scale out by adding more nodes to supply processing units and storage. Cassandra, HBase, HDFS and MongoDB are examples of large-data storage platforms that leverage shared-nothing designs to address storage limits by horizontally expanding out to new nodes, allowing for huge data expansion. The following are some of the characterized criteria used to compare the aforementioned big-data storage systems: server operating systems, methods of partitioning, data scheme, concurrency, programming languages and others, as seen in Table 3. The following are the 3 kinds of data models for storage that can be broadly grouped: (I) A file system such as HDFS. Data is saved schemaless in HDFS and taken logically at processing time based on the processing application's requirements, a technique refers to as "Schema-on-Reading". (II) Document-based, example, MongoDB; and (III) Column-based schema, example, Cassandra and Hbase [31].

Table 3. Storage technologies for big data.

Features	**Cassandra**	**Hadoop Hive**	**MongoDB**	**HBase**
OS server	FreeBSD, Linux, OS X, Windows	All operating systems that have a "Java virtual	Linux, OS X, Windows, Solaris,	Windows, Unix, Linux,
Model for storing data	Column-based	File-system	Document-based	Column based
key-value for MapReduce	Yes	Yes	Yes	Yes
Concurrency	Yes	Yes	Yes	Yes
Capabilities for in-memory	Yes	N/A	Yes	Yes
Theorem of CAP	Consistency Partition tolerance	Consistency Partition tolerance	Availability Partition tolerance	Consistency Availability
Programming languages supported	"C#, C++, Clojure, Erlang, Go, Haskell, Java, Node.js, Perl, PHP, Python, Ruby, Scala"	C++ Java PHP Python	"C#, C++, Clojure, Erlang, Go, Haskell, Java, Node.js, Perl, PHP, Python, Ruby, Scala"	"C, C#, C++, Groovy, Java, PHP, Python, Scala"
Concept consistency	Eventual Consistency Immediate Consistency	Eventual Consistency	Eventual Consistency Immediate Consistency	Immediate Consistency

Methods of APIs and other access	ODBC and JDBC	JDBC, ODBC, Thrift	Proprietary protocol using	ODBC and JDBC
Description	Large volumes of structured data can be managed with a distributed database.	Data warehouse software for querying and managing large distributed datasets, based on	One of the most popular document storage options	Open-source, networked, versioned and column-oriented database
Partitioning methods	Key partitioning	Shading	Shading	Key partitioning
Data scheme	Relational DBMS uses Amazon DynamoDB	Relational DBMS Schema-on-Reading	Schema-free	Relational DBMS uses Google Bigtable
Replication	Masterless-ring	Selectable replication factor	Master/slave Replication	Master/slave Replication
Base code	Java	Java	C++	Java

In-memory data processing has recently gained popularity in developing technologies, with RAM and flash memory replacing slower drives. As a result, we may differentiate large-data storage solutions based on their ability to handle data in memory, which is especially important for essential real-time applications. Representatives of this mechanism include MongoDB, Cassandra and HBase. As a result, in our research, we adopted Cassandra because of its superior query performance and always-on features, as well as its distributed capability for real-time applications. Cassandra has a masterless "ring" architecture, which has several advantages over traditional master-slave topologies. As a result, each node in a cluster is regarded evenly, so quorum can be achieved by using a majority of nodes.

6. REAL-TIME PROCESSING OF STATE-OF-THE-ART ARCHITECTURE

Lambda and Kappa are two real-time processing architectures that are presented in this study. We evaluated them using their specifications and came up with a stronger solution that meets the real-time requirements specified previously.

Batch processing, as shown in the literature, performs processing on huge datasets with great throughput and efficiency, but it usually takes a long time. It could take several hours, which is far too much latency for almost any current application to provide live results. Stream processing, on the other hand, works with the most recent records that enter the system, allowing for quick processing and near-real-time results, but at the cost of being less precise than batch processing. Nathan Marz proposed the Lambda architecture [40], which combines both types of processing to gain their benefits in one architecture, providing real-time results and correct perspectives with low latency and high throughput with fault tolerance. This architecture is made up of 3 levels; namely, the batch layer, the speed layer and the serving layer (depicted in figure 8).

The batch layer generates batch views and keeps track of the master copy of the dataset. The serving layer incorporates the findings from the batch and speed layers. To compensate for the significant latency of the service layer updates, the speed layer only processes the most current data. The batch and speed processing layers are on the same level in the architecture. This means that the fresh raw data is provided to both of them at the same time. In the meantime, the serving layer is located above as seen in Figure 8. However, due to its intricacy, this architecture has significant drawbacks as well as some criticism. This architecture necessitates the integration of numerous systems and technologies, which adds to the process's complexity. In addition, because there are two processing levels, distinct processing codes must be maintained and kept in sync to provide views to the serving layer. This also highlights the fact that such routines might be written in a variety of programming languages. Finally, the serving layer is fed by two separate layers whose data, aside from the batch layer's pre-stored data, will be identical, implying that data, information and logic will be duplicated.

Kappa architecture is a lambda architecture simplification. It is a software architectural pattern designed by Jay Kreps in 2014 based on his LinkedIn experience [3]. With the exception that all data travels over a single conduit, the stream layer, the Kappa design delivers the same benefits as the Lambda architecture. Data is appended to a unified, distributed and fault-tolerant log and its status is

only updated when such appends occur. This allows for view recalculations or recomputations. To do so, the data is streamed back in from the beginning. To avoid losing the prior computation, a parallel task is started, allowing two computations to be done at the same time. Following the completion of the second computation, the developer must decide whether to keep both, combine them or remove the prior one and keep the last one if it exceeds expectations. The Kappa architecture, which is made up of two levels, is depicted in Figure 9. The results are queried using the serving layer and the stream processing jobs are executed using the stream processing layer.

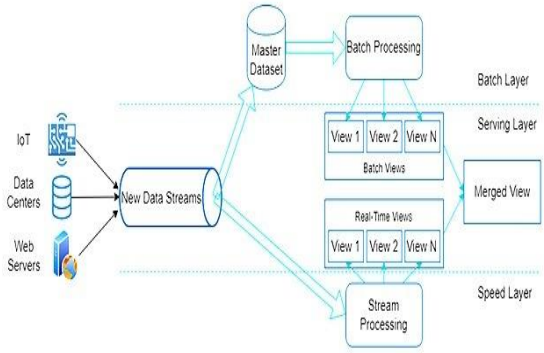

Figure 8. Lambda architecture.

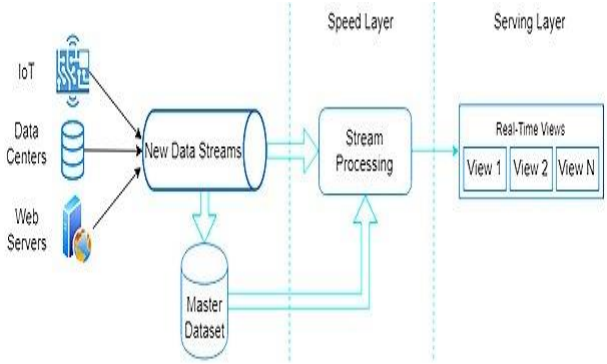

Figure 9. Kappa architecture.

Table 4 shows a brief comparison of the two architectures, Lambda and Kappa, as previously mentioned, using certain criteria.

Table 4. Comparison between lambda and kappa architecture.

Features	**Lambda**	**Kappa**
Real-time	Isn't accurate	Correct (accurate)
Fault tolerance	Yes	Yes
Architecture	Immutable	Immutable
Scalability	Yes	Yes
Permanent storage	Yes	No
Guarantees processing	Yes, in batch approximate in streaming	Exactly-once with consistency
Re-processing paradigm	During each batch cycle	Only when there is a code update
Data processing	Real-time and batch	Real-time
Layers	Batch, serving and real-time layers	Stream processing and serving layers

The lambda architecture is one of two architectures used in big-data systems and it allows for simultaneous processing of enormous datasets as well as continuous real-time access to them. The goal behind this architecture is to build two independent processes, one for batch data processing and the other for real-time data access. The batch layer performs calculations on the whole data collection. It takes time, but the data returned is complete and of good quality. The dataset in the batch layer is believed to be intact [41]. You can maintain data consistency and access to past data in this way. Incoming data is processed in real time by the real-time layer. The speed with which this layer's data may be accessed correlates to the prospect of speedier information retrieval. Unfortunately, due to a lack of historical data, not all computations can be performed [41]; hence, as seen in Table 4, real time “isn't accurate” in the lambda architecture; however, the kappa architecture can overcome this setback, providing accurate real time processing. Both architectures are fault-tolerant, immutable, scalable and have guaranteed processing ability, as can be seen in Table 4. The requirement to sustain two distinct applications: one for supporting the batch layer and the other for supporting the real-time layer, is the biggest and most frequently noted downside of the lambda architecture. Because the tools used in each layer differ, it's difficult to pick one that can serve two objectives. Unfortunately, maintaining this design is more difficult and costly [41]-[42]. The kappa architecture maintains a single pipeline; hence, it is easier to manage. Therefore, in our study, we adopted the kappa architecture as a result of its numerous advantages over the lambda architecture. However, the kappa architecture had no permanent storage, as seen in Table 4, but in our proposed framework, we introduced the permanent-storage layer.

7. PROPOSED FRAMEWORK

Taking into consideration the unique peculiarities of spatiotemporal big data, as well as overviews of stream-processing platforms, distributed message queuing systems and big-data storage technologies, the two state-of-the-art real-time architectures presented in this study have numerous advantages and disadvantages. Based on the findings of the literature, we suggested the open-source framework depicted in Figure 10, which has a unique set of properties, the most notable of which is its capacity to analyze massive amounts of spatiotemporal data in real time at high speed. It also allows an unlimited number of users to create new and unique features as well as make various reforms. The proposed framework closely resembles the kappa architecture, which provides more benefits than the lambda design. However, because the kappa architecture lacks a storage layer, we included one in our proposed framework.

In our proposed framework, there is the data source, from which is where the spatiotemporal datasets are obtained. Sensor networks, online traffic, social media, video streams and other sources could all yield distinct dataset structures, resulting in a large problem known as heterogeneous data [38]. This brings in the data ingestion layer which streams data from various upstream applications and fed to real-time downstream applications using distributed queueing management technologies. In our proposed framework we adopted Kafka as a result of its numerous advantages over other message-queueing systems. Kafka is highly scalable and most importantly can handle the challenge of heterogeneous data which is also a major problem with spatiotemporal datasets. Kafka is a data-transit technology rather than a data processing system. This distinguishes it from the competition in stream processing with high throughputs. The spatiotemporal dataset which is produced from the various data sources is transformed and filtered by Kafka and a common format is produced for either storage or immediate computations as proposed by our framework. We must first install the Kafka cluster, then launch Zookeeper and the Kafka server to get Kafka up and running. Zookeeper monitors the state of Kafka cluster nodes and keeps track of Kafka topics, partitions and other data. Kafka provides an inbuilt *KafkaProducer<k, v>* class that uses the serialization process to store streaming data in a user-defined format (e.g. *CustomObject*). It is the conversion of a specified data type into byte format [42]. The configuration properties file is used to create the Kafka producer. The topic name and CustomObject are the key-value pair. The syntax is: "*Producer<String, CustomObject> producer =new KafkaProducer<String, CustomObject> (configProperties);*". The *KafkaConsumer<k, v>* class reads and deserializes the streaming data from the Kafka producer. The process of transforming a byte format to the desired format is known as derealization [42]. The ingestion layer is very important in real-time spatiotemporal data analysis, as the cleansing and preprocessing of data is carried out here. The next layer is the real-time processing spatiotemporal data layer, which is focused on real time data processing with low latency. In our proposed framework, Flink was adopted as a result of its true real-time processing ability. Hence, our major goal is to propose a true real-time processing framework as the spatiotemporal datasets are fed into the system for prompt and immediate results. Flink has a very strong unique feature that makes it tall among other stream-processing engines or computational engines, which is CEP [37]. CEP systems assess queries against uninterrupted streams of events to find trends [29]. CEP's goal is to analyze data as it enters our system, so we don't have to keep it somewhere unless it's necessary. Also, the goal of CEP is to analyze and react to streams of events. Machine learning which is also part of our proposed framework at the processing layer is responsible for real-time prediction. Our framework has been designed to incorporate all these functionalities. At the end of the real-time processing layer, the output is sent to the storage layer, where we adopted the Cassandra as a result of its distributed ability. Cassandra has many benefits, such as a completely decentralized design with no single point of failure, promising linear scalability, great write performance and configurable data consistency levels within queries. A ring of nodes organizes the Cassandra cluster. Each of these nodes is in charge of storing a portion of the data. The hash keyspace is divided by the total number of *tokens* selected for the database to provide an equal data distribution inside the ring. A random subset of potential primary hash key values is connected with a node based on the number of *tokens* issued to it. This subset of data becomes the responsibility of the node for the entire database. Each node in the ring usually has the same quantity of *tokens*. Cassandra replicates data on other nodes in the ring to ensure high availability. The *Replication Factor (RF)* determines the number of replicas. This means that each node in a cluster of *N* nodes will store a piece of the keyspace equal to *RF=N* [43]. The visualization layer's primary responsibility is to transmit the final

data and outcomes in streaming mode to the user. If all processes are completed correctly, this layer can respond quickly.

7.1 Comparison of the Proposed Framework

The proposed framework has been designed to tackle various issues with both lambda and kappa systems. Lambda enables clients to have the most up-to-date vision. However, business logic is performed at both layer levels, two distinct sources of the same data are required to feed the next layer and this design requires many frameworks to set up. Kappa architecture was established as a result of the complexities of lambda architecture. Unlike lambda, kappa evolves to be more focused on data processing, even though it does not support permanent data storage. This architecture is less complicated than lambda and allows the user to select which implementation composers to use. However, kappa is not a magical formula that can solve all of the big spatiotemporal-data problems. Furthermore, instead of addressing data-quality issues or data-analysis outcomes, these two architectures focus on balancing throughput and latency to handle performance challenges. The kappa architectural principle underpins our proposed framework. It's a streaming data-processing approach that allows for long-term data storage by treating all incoming data as streaming data. The suggested framework can deliver actual real-time processing using Flink and machine learning. Flink is a fault-tolerant distributed real-time computing system with many other advantages, as detailed in the previous sections. By efficiently combining and expanding sophisticated real-time computations in a computer cluster, Flink enables the reliable processing of infinite streams of data. In another comparison, the proposed framework was also created to address specific aspects that are strongly linked to spatiotemporal big data in stream processing. Some of these characteristics are scalability (Sc), data analytics (DA), multiple event types (MET), prediction tools (PT), data storage (DS), real-time (Rt), performance evaluation (PE) and stream processing (SP). Table 5 compares our novel framework with others and our framework has all the capabilities, which are challenging to stream processing for big spatiotemporal data; hence, our proposed framework can handle all these characteristics.

From Table 5, the check-marks (√) indicated that the existing framework from previous works can perform the important functions (Sc, SP, DA, MET, PT, Rt and PE) as regards big spatiotemporal data in stream-processing frameworks, while the check-marks (-) indicate that the framework cannot perform the earlier listed functions, since the authors did not incorporate them in their frameworks. As discussed earlier in Section 2, these characteristics or parameters are very important for real-time stream processing with big spatiotemporal data, making the system more robust and relevant, since it has all the required characteristics; hence, in our proposed framework, we made the provisions to accommodate all these characteristics.

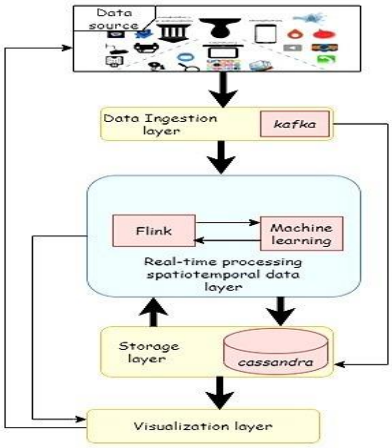

Figure 10. Proposed framework.

Table 5. Comparison of our proposed framework with others.

	Sc	SP	DA	DS	MET	PT	Rt	PE
Corral-Plaza et al., [38]	√	√	√	√	√	-	√	√
Carcillo et al., [33]	√	√	√	√	-	√	-	-
Amini et al., [44]	√	√	√	√	-	√	√	-
D'silva et al., [45]	√	√	√	√	√	-	√	√
Jung et al., [46]	√	-	√	-	-	-	-	√
Montori et al., [47]	-	-	√	√	-	-	-	-
Santos et al., [48]	-	-	√	√	-	√	√	-
Our proposal	√	√	√	√	√	√	√	√

8. CONCLUSIONS

We have developed a novel framework in this study that may be used in a variety of spatiotemporal big-data scenarios. Its key innovative advantage is the capacity to automatically handle and analyze spatiotemporal data regardless of structure. The inclusion and utilization of (1) Kafka as process

streams of spatiotemporal data sources as they occur; (2) Apache Flink as the computational layer and (3) Apache Cassandra as the storage layer for real-time distributed storage have benefited this framework. The study's major purpose is to present a true real-time processing paradigm using Flink and machine learning. In our proposed design, we suggested emphasizing the real-time processing layer and we did our best to optimize it with Flink and machine learning. The advantages of the technologies employed, as well as the advantages of kappa design after it was compared with the lambda architecture, where the key inspirations for this innovative building were obtained, are presented. The study highlighted stream-processing technologies, queueing-messaging systems and big-data storage technologies and presented their comparison for a better choice. Looking at the best tools and their advantages over others, the study proposed a novel true real time spatiotemporal data stream-processing framework. Hence, an important framework for processing and analysing spatiotemporal data from multiple sources with irregular shapes has been proposed, so that researchers can focus on data analysis instead of worrying about the data sources' structure. The following stage is to validate and assess its performance. The vast majority of research, including this one, has certain limitations. However, until the validation process is completed, we won't be able to examine its shortcomings.

ACKNOWLEDGEMENTS

The authors would like to thank everyone, just everyone!

REFERENCES

[1] B. R. Hiraman, M. C. Viresh and C. K. Abhijeet, "A Study of Apache Kafka in Big Data Stream Processing," Proc. of the Int. Conf. on Information, Communication, Engineering and Technology (ICICET 2018), pp. 1–3, DOI: 10.1109/ICICET.2018.8533771, 2018.

[2] J. Manyika, M. Chui Brown, B. B. J., R. Dobbs, C. Roxburgh and A. Hung Byers, "Big Data: The Next Frontier for Innovation, Competition and Productivity," McKinsey Global Institute, no. June, p. 156, [Online], Available: https://bigdatawg.nist.gov/pdf/MGI_big_data_full_report.pdf, 2011.

[3] S. Ounacer, M. Amine, S. Ardchir, A. Daif and M. Azouazi, "A New Architecture for Real Time Data Stream Processing," International Journal of Advanced Computer Science and Applications, vol. 8, no. 11, pp. 44–51, DOI: 10.14569/ijacsa.2017.081106, 2017.

[4] F. Pivec, "The Global Information Technology Report 2003–2004," Organizacija Znanja, vol. 8, no. 4, pp. 203-206, DOI:10.3359/oz0304203, 2003.

[5] S. Nadal et al., "A Software Reference Architecture for Semantic-aware Big Data Systems," Information and Software Technology, vol. 90, pp. 75–92, DOI: 10.1016/j.infsof.2017.06.001, 2017.

[6] A. Hamdi, K. Shaban, A. Erradi et al., "Spatiotemporal Data Mining: A Survey on Challenges and Open Problems," Artificial Intelligence Review, no. 0123456789, DOI: 10.48550/arXiv.2103.17128, 2021.

[7] N. Khan, et al., "The 10 Vs, Issues and Challenges of Big Data," Proc. of the ACM Int. Conf., no. March, pp. 52–56, DOI: 10.1145/3206157.3206166, 2018.

[8] R. L. Villars, C. W. Olofson and M. Eastwood, "Big Data: What It is and Why You Should Care," IDC White Paper, pp. 7–8, 2011, [Online], Available: http://www.tracemyflows.com/ uploads/big_data/IDC_ AMD_Big_Data_Whitepaper.pdf, 2011.

[9] N. Elgendy and A. Elragal, "Big Data Analytics: A Literature Review," Journal of Management Analytics, vol. 2, no. 3, pp. 214–227, 2014.

[10] C. L. Philip Chen and C. Y. Zhang, "Data-intensive Applications, Challenges, Techniques and Technologies: A Survey on Big Data," Information Sciences, vol. 275, pp. 314–347, 2014.

[11] S. Salehian and Y. Yan, "Comparison of Spark Resource Managers and Distributed File Systems," Proc. of the IEEE Int. Conf. on Big Data and Cloud Computing (BDCloud), Social Computing and Networking (SocialCom), Sustainable Computing and Communications (SustainCom) (BDCloud-SocialCom-SustainCom), pp. 567–572, DOI: 10.1109/BDCloud-SocialCom-SustainCom.2016.88, 2016.

[12] J. Jo and K. W. Lee, "MapReduce-based D-ELT Framework to Address the Challenges of Geospatial Big Data," ISPRS Int. Journal of Geo-information, vol. 8, no. 11, DOI: 10.3390/ijgi8110475, 2019.

[13] J. Kang, L. Fang, S. Li and X. Wang, "Parallel Cellular Automata Markov Model for Land Use Change Prediction over MapReduce Framework," ISPRS Int. Journal of Geo-Information, vol. 8, no. 10, DOI: 10.3390/ijgi8100454, 2019.

[14] D. Glushkova, P. Jovanovic and A. Abelló, "MapReduce Performance Model for Hadoop 2.x," Information Systems, vol. 79, pp. 32–43, DOI: 10.1016/j.is.2017.11.006, 2019.

[15] I. A. T. Hashem et al., "MapReduce Scheduling Algorithms: A Review," The Journal of Supercomputing, vol. 76, pp. 4915–4945, 2020.

[16] F. Li, J. Chen and Z. Wang, "Wireless MapReduce Distributed Computing, " IEEE Transactions on

Information Theory, vol. 65, no. 10, pp. 6101–6114, DOI: 10.1109/TIT.2019.2924621, 2019.

[17] M. Bendre and R. Manthalkar, "Time Series Decomposition and Predictive Analytics Using MapReduce Framework," Expert Systems with Applications, vol. 116, pp. 108–120, 2019.

[18] S. Heidari, M. Alborzi, R. Radfar et al., "Big Data Clustering with Varied Density Based on MapReduce," Journal of Big Data, vol. 6, no. 1, DOI: 10.1186/s40537-019-0236-x, 2019.

[19] N. Maleki, A. M. Rahmani and M. Conti, "MapReduce: An Infrastructure Review and Research Insights," The Journal of Supercomputing, vol. 75, pp. 6934–7002, 2019.

[20] S. Wang, Y. Zhong and E. Wang, "An Integrated GIS Platform Architecture for Spatiotemporal Big Data," Future Generation Comp. Sys., vol. 94, pp. 160–172, DOI: 10.1016/j.future.2018.10.034, 2019.

[21] M. M. Alam, L. Torgo and A. Bifet, "A Survey on Spatio-temporal Data Analytics Systems," ACM Computing Surveys, pp. 1–37, DOI: 10.1145/3507904, 2022.

[22] Apache, "Apache Hadoop: An Open-source Distributed Processing Framework," [Online], Available: https://hadoop.apache.org/, 2020.

[23] A. Davoudian, L. Chen and M. Liu, "A Survey on NoSQL Stores," ACM Computing Surveys, vol. 51, no. 2, DOI: 10.1145/3158661, 2018.

[24] M. Zaharia et al., "Apache Spark: A Unified Engine for Big Data Processing," Communications of the ACM, vol. 59, no. 11, pp. 56–65, DOI: 10.1145/2934664, 2016.

[25] I. Yaqoob, I. A. T. Hashem, A. Ahmed, S. M. A. Kazmi and C. S. Hong, "Internet of Things Forensics: Recent Advances, Taxonomy, Requirements and Open Challenges," Future Generation Computer Systems, vol. 92, no. May 2018, pp. 265–275, DOI: 10.1016/j.future.2018.09.058, 2019.

[26] E. Ahmed et al., "The Role of Big Data Analytics in Internet of Things," Computer Networks, vol. 129, pp. 459–471, DOI: 10.1016/j.comnet.2017.06.013, 2017.

[27] S. Henning and W. Hasselbring, "How to Measure Scalability of Distributed Stream Processing Engines?" Proc. of Companion of the ACM/SPEC Int. Conf. on Performance Engineering (ICPE 2021), pp. 85–88, DOI: 10.1145/3447545.3451190, 2021.

[28] K. Kallas, F. Niksic, C. Stanford and R. Alur, "Stream Processing with Dependency-guided Synchronization," Proc. of the 27th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming (PPoPP '22), pp. 1-16, DOI: 10.1145/3503221.3508413, Seoul, Republic of Korea, 2022.

[29] N. Giatrakos, E. Alevizos, A. Artikis, A. Deligiannakis and M. Garofalakis, "Complex Event Recognition in the Big Data Era: A Survey," VLDB Journal, vol. 29, no. 1, pp. 313–352, 2020.

[30] T. Li, Z. Xu, J. Tang and Y. Wang, "Model-free Control for Distributed Stream Data Processing Using Deep Reinforcement Learning," Proc. of the VLDB Endowment, vol. 11, no. 6, pp. 705–718, 2018.

[31] R. Sahal, J. G. Breslin and M. I. Ali, "Big Data and Stream Processing Platforms for Industry 4.0 Requirements Mapping for a Predictive Maintenance Use Case," Journal of Manufacturing Systems, vol. 54, no. November 2019, pp. 138–151, DOI: 10.1016/j.jmsy.2019.11.004, 2020,

[32] D. P. Carazo, Evaluation and Deployment of Big Data Technologies on a NIDS Evaluación y Despliegue de Tecnologías Big Data Sobre un NIDS, M.Sc. Thesis, Master in Data Science, Universidad Internacional Menendez Pelayo, 2019.

[33] F. Carcillo, A. Dal Pozzolo, Y. A. Le Borgne, O. Caelen, Y. Mazzer and G. Bontempi, "SCARFF: A Scalable Framework for Streaming Credit Card Fraud Detection with Spark," Information Fusion, vol. 41, pp. 182–194, DOI: 10.1016/j.inffus.2017.09.005, 2018.

[34] H. Herodotou, Y. Chen and J. Lu, "A Survey on Automatic Parameter Tuning for Big Data Processing Systems," ACM Computing Surveys, vol. 53, no. 2, DOI: 10.1145/3381027, 2020.

[35] M. Dias de Assunção, A. da Silva Veith and R. Buyya, "Distributed Data Stream Processing and Edge Computing: A Survey on Resource Elasticity and Future Directions," Journal of Network and Computer Applications, vol. 103, no. July 2017, pp. 1–17, DOI: 10.1016/j.jnca.2017.12.001, 2018.

[36] A. Batyuk and V. Voityshyn, "Apache Storm Based on Topology for Real-time Processing of Streaming Data from Social Networks," Proc. of the 1st IEEE Int. Conf. on Data Stream Mining and Processing (DSMP 2016), no. August, pp. 345–349, DOI: 10.1109/DSMP.2016.7583573, 2016.

[37] B. Zhao, H. Van Der Aa, T. T. Nguyen, Q. V. H. Nguyen and M. Weidlich, "EIRES: Efficient Integration of Remote Data in Event Stream Processing," Proc. of the ACM SIGMOD Int. Conf. on Management of Data, no. i, pp. 2128–2141, DOI: 10.1145/3448016.3457304, 2021.

[38] D. Corral-Plaza, I. Medina-Bulo, G. Ortiz and J. Boubeta-Puig, "A Stream Processing Architecture for Heterogeneous Data Sources in the Internet of Things," Computer Standards and Interfaces, vol. 70, no. June 2019, p. 103426, DOI: 10.1016/j.csi.2020.103426, , 2020.

[39] N. Tantalaki, S. Souravlas and M. Roumeliotis, "A Review on Big Data Real-time Stream Processing and Its Scheduling Techniques," International Journal of Parallel, Emergent and Distributed Systems, vol. 35, no. 5, pp. 571–601, DOI: 10.1080/17445760.2019.1585848, 2020.

[40] N. Marz, Big Data: Principles and Best Practices of Scalable Realtime Data Systems, ISBN:978-1-61729-034-3, [S.l.]: O'Reilly Media, 2013.

[41] J. Bobulski and M. Kubanek, "Data Model for Bigdata System for Multimedia," Proc. of the ACM Int. Conf. Proceeding Series, vol. PartF16898, pp. 12–17, DOI: 10.1145/3449365.3449368, 2021.

[42] A. Bandi and J. A. Hurtado, "Big Data Streaming Architecture for Edge Computing Using Kafka and Rockset," Proc. of the 5th Int. Conf. on Computing Methodologies and Communication (ICCMC 2021), no. Iccmc, pp. 323–329, DOI: 10.1109/ICCMC51019.2021.9418466, 2021.

[43] S. Dipietro, G. Casale and G. Serazzi, "A Queueing Network Model for Performance Prediction of Apache Cassandra," Proc. of the 10th EAI Int. Conf. on Performance Evaluation Methodologies and Tools (ValueTools 2016), pp. 186–193, DOI: 10.4108/eai.25-10-2016.2266606, 2017.

[44] S. Amini, I. Gerostathopoulos and C. Prehofer, "Big Data Analytics Architecture for Real-time Traffic Control," Proc. of the 5th IEEE Int. Conf. on Models and Technologies for Intelligent Transportation Systems (MT-ITS), pp. 710–715, DOI: 10.1109/MTITS.2017.8005605, 2017.

[45] G. M. D'silva, A. Khan, Gaurav and S. Bari, "Real-time Processing of IoT Events with Historic Data Using Apache Kafka and Apache Spark with Dashing Framework," Proc. of the 2nd IEEE Int. Conf. on Recent Trends in Electronics, Information Communication Technology (RTEICT), pp. 1804–1809, DOI: 10.1109/RTEICT.2017.8256910, 2017.

[46] H. S. Jung, C. S. Yoon, Y. W. Lee, J. W. Park and C. H. Yun, "Cloud Computing Platform Based Real-time Processing for Stream Reasoning," Proc. of the 6th Int. Conf. on Future Generation Communication Technologies (FGCT 2017), pp. 37–41, DOI: 10.1109/FGCT.2017.8103400, 2017.

[47] F. Montori, L. Bedogni and L. Bononi, "A Collaborative Internet of Things Architecture for Smart Cities and Environmental Monitoring," IEEE Internet of Things Journal, vol. 5, no. 2, pp. 592–605, DOI: 10.1109/JIOT.2017.2720855, 2018.

[48] P. M. Santos et al., "PortoLivingLab: An IoT-based Sensing Platform for Smart Cities," IEEE Internet of Things Journal, vol. 5, no. 2, pp. 523–532, DOI: 10.1109/JIOT.2018.2791522, 2018.

ملخص البحث:

تُعدّ القدرة على تفسير سيول البيانات المؤقّتة حيّزيّاً أمراً حاسماً بالنسبة للعديد من الأنظمة. ومع ذلك، فإنّ معالجة كمّيات هائلة من البيانات المؤقّتة حيّزيّاً من مصادر متنوعة، مثل التّواصل عبر الإنترنت، ومنصّات التّواصل الاجتماعي، وشبكات المجسّات وغيرها، تشكّل تحدّياً ملحوظاً. لذلك، فإنّ الهدف الأساسي من هذا البحث هو إيجاد إطارٍ لمعالجة وتحليل البيانات المؤقّتة حيّزيّاً من مصادر متعدّدة وبأشكالٍ غير منتظمة، بحيث يتمكّن الباحثون من التّركيز على تحليل البيانات بدلاً من الاهتمام ببنية مصادر البيانات.

نقترح في هذا البحث نموذجاً جديداً لمعالجة سيول البيانات المؤقّتة حيّزيّاً، يمكّن من معالجة البيانات بسرعةٍ عالية وبتأخُّر طفيفٍ عن الزّمن الحقيقي، مع أخذ الاعتبارات آنفة الذّكر بعين الاعتبار. كذلك نجري مقارنةً بين النّموذج المقترح وعددٍ من النّماذج المتبنّاة في دراسات سابقة تتعلّق بمعالجة البيانات الضّخمة في الزّمن الحقيقي. هذا الى جانب نظرة شاملة على تقنيات المصادر المفتوحة المستخدمة في معالجة سيول البيانات في الزّمن الحقيقي.

يتميّز النموذج المقترح بأنّه يدمج بين (أباتشي كافكا) لاستقبال البيانات من مصادرها، و (أباتشي فلينك) لمعالجة سيول البيانات، وتعلُّم الآلة للتّوقّعات في الزّمن الحقيقي، و (أباتشي كاساندْرا) في طبقة التّخزين من أجل التّخزين الموزّع في الزّمن الحقيقي. وقد تمّت مقارنة النّموذج المقترح مع عددٍ من النّماذج الأخرى المستخدمة لمعالجة سيول البيانات في الزّمن الحقيقي، وذلك بناءً على مجموعة من الخصائص، مثل [إمكانية التّوسيع، وأدوات التّوقُّع، وتحليل البيانات، وأنواع الأحداث المتعدّدة، وتخزين البيانات، والزّمن الحقيقي، وتقييم الأداء في معالجة سيول البيانات]. وقد أثبت النّموذج المقترح تفوُّقاً على النّماذج الأخرى وبرهن على فعاليته في التّعامل مع جميع المسائل ذات العلاقة.