Future Generation Computer Systems 113 (2020) 207–217

Contents lists available at ScienceDirect

Future Generation Computer Systems

journal homepage: www.elsevier.com/locate/fgcs

Performance evaluation of real-time stream processing systems for Internet of Things applications

Vikash [*], Lalita Mishra, Shirshu Varma

Department of Information Technology, Indian Institute of Information Technology, Allahabad, Uttar Pradesh 211015, India

A R T I C L E   I N F O

Article history:
Received 20 January 2020
Received in revised form 22 June 2020
Accepted 6 July 2020
Available online 8 July 2020

Keywords:
Wireless sensor networks
Internet of Things
Pervasive computing
Middleware
Stream processing
Real-time processing

A B S T R A C T

In the current scenario, IoT is an ideal and novel technology, which fulfills the needs of most of the commercial, non-commercial, government, and private organizations by its real-time supportive nature and characteristics. However, real-time processing itself a very critical research topic. But, most of the IoT applications are empowered by real-time data processing. Thus, it become a vital part of IoT.

In this work, we proposed a four-layer infrastructure for IoT along with stream processing. Further, we use stream processing techniques along with IoT infrastructure for applications and analyze the performance of stream processing techniques for IoT applications. Also, we compare and find the five most suitable distributed stream processing systems for IoT, based on its performance and characteristics. We use two benchmark applications to evaluate the performance of distributed stream processing systems against response time, throughput, jitter, and scalability. Based on that, we suggest the adapted solution for IoT applications.

We evaluate the performance with peak stream rates from 100k to 1M along with the various frequencies of benchmark applications. Further, on the basis of results, we conclude that Apache NiFi is the most suitable solution for IoT applications.

1. Introduction

The Internet of Things (IoT) has a significant impact on the world of digitization towards pervasiveness. Now, it is playing an inevitable role in our lives. Many companies and government organizations are already using IoT for better and automated services with the aim of pervasiveness. According to Ericcson (one of the leading telecommunication companies), there will be around 29 billion connected devices by 2022; among them, 18 billion will only be the IoT devices, and it is also predicted that 70% of the IoT devices will use cellular technology. McKinsey (an American management consulting firm) predicts that the economic impact of IoT could reach $3.9 trillion to $11.9 trillion per year by 2025.

These devices generate an immense amount of continuous data. However, this whole ecosystem, along with its heterogeneous nature is of no use without analyzing data in real-time, and stream processing facilitate the best technologies for real-time data analytics to produce desired instructions. So, the system can intimate the desired instructions within the time frame.

Stream processing is a technology, that allows its applications to act on real-time data (by collecting, integrating, analyzing, and visualizing), while the data is being generated [1]. In other words, stream processing technology processes a large amount of data in motion, which belongs to multiple sources. In IoT, stream processing systems are used as a part of middleware to process continuous data streams from IoT devices to get insights from continuous real-time data. Moreover, stream processing get insights from real-time data with high throughput and low latency for the massive amount of continuous data generated from IoT devices. However, stream processing systems need to work in parallel with the distributed environment to fulfill the desired needs [2]. To evaluate the real-time data streaming system, we need to choose the most opportune distributed stream processing system (DSPS), which fulfills the desired requirements of IoT applications. Further, we test the chosen technology and evaluate the performance of IoT applications. In this paper, we have chosen five streaming platforms and implement each of them to test their suitability for IoT applications. Further, we describe the essential characteristics of the processing platform for IoT applications, which should not be missed while we are applying stream processing in real-time.

1.1. Important characteristics of IoT applications

Nowadays, there are many applications, which works with IoT. These applications generate a massive amount of data continuously, which needs to process in real-time to produce desired

* Corresponding author.
E-mail addresses: rsi2016503@iiita.ac.in (Vikash), rsi2018502@iiita.ac.in (L. Mishra), shirshu@iiita.ac.in (S. Varma).

https://doi.org/10.1016/j.future.2020.07.012
0167-739X/© 2020 Elsevier B.V. All rights reserved.

results. Further, there are several characteristics, which needs to be adapted by stream processing for IoT applications, as follows:

1.1.1. *Real-time*

It is characterized as the time, at which the event occurs, and recording or reporting of the event is approximately in no time or timeliness. In terms of the data processing systems, the middleware system can receive constantly altering data from various sources, like data related to health monitoring, smart home, etc. and process the data stream [3] rapidly, which can control the source of data by providing the control instructions.

1.1.2. *Automation*

It is defined as a system or method, which is used to control processes without human intervention. In terms of IoT, smart physical devices that connect with other physical devices can operate automatically without any input from any human operators, which is an essential characteristic of IoT applications. International Electrotechnical Commission (IEC) is a Switzerland based standard organization, which defines IEC 61499 [4] standard for distributed automation.

1.1.3. *Distributed*

It is a nature of the system in which two or more connected devices can perform the task independently or jointly. IoT works in a pervasive environment, so the processing system should be able to work in a distributed manner. In this direction, Haim et al. [5] presented the abstract view of all the standards related to all the activities of the distributed nature of the system.

1.1.4. *Security*

It defines the premeasured aspect in the protection of digital information and physical devices against external and internal attacks along with accidental and malicious threats. Security is a necessity for all types of enterprises and both public and private organizations of all sizes. In this direction, Disterer [6] presents International Organization for Standardization (ISO)/IEC series of standards related to information security to provide an efficient solution.

1.1.5. *Reliability*

It is the quality of the correctness of information from distributed sources and defines the dependability and trustworthiness of the sources. In terms of IoT, it defines a set of promises committed at the time of system development. Moreover, it defines as the degree of system, components related to specific functions with specific conditions over real-time. ISO [7] presents the general principles related to the reliability of any structure.

1.1.6. *Scalability*

The term scale refers to "the capability of the system", and scalability defines the ability of the system to adapt and inherit the new objectives in terms of system enhancement to fulfill the increasing demands. As IoT growing continuously, so we need a system that fulfills the current demand and should be extendable according to future demands. In 2017, ISO [8] presents the standard for scalability along with the convergence of heterogeneous networks, which is an efficient solution for IoT applications.

1.1.7. *Fault-tolerant*

Generally, the term fault-tolerant define the ability of the system that, if any operation failure occurs at any stage, then the system will navigate to a safe condition. It defines the ability of the system to allow failures or malfunctions; that can be provided by the software, hardware, and combination of both. IEC [9] defines the standards for fault-tolerant and safety for various electronic devices and related systems.

1.1.8. *Statefulness*

It defines the consistency among the different states of the system. IoT applications work in a distributed environment. It means that the state is shared among multiple states. Thus, the results that belong to the past event can influence the processing of the current event. In 2019, Internet Engineering Task Force (IETF) [10] standardized statefulness among the various states of the system.

These characteristics are essential for IoT applications towards an efficient solution. Thus, stream processing systems need to follow these characteristics to facilitate an efficient solution for IoT applications. Further, we define how the stream processing system meets with IoT applications.

1.2. *Distributed stream processing meets IoT*

In the current scenario, IoT applications generate continuous and massive amounts of data. So, it comes in the category of Big Data to process the massive amount of data. Traditionally, the methods used to process the big data involve MapReduce jobs that process the data in the batch processing technique, which is not a suitable solution for real-time scenarios [11] like Hadoop Distributed File System (HDFS). However, stream processing technologies enable applications to process the data in real-time as soon as it is available, without storing it. Thus, stream processing automatically becomes a suitable solution for IoT applications.

IoT applications work in a distributed environment and generate continuously massive amount of data, which needs to process in real-time. DSPS technologies are developed to process real-time data streams without HDFS. Nowadays, this technology is prevalent for real-time processing and has excellent growth in its user base.

1.3. *Our main contribution*

The main insight of this work is to propose a DSPS enabling infrastructure for IoT applications, to enhance the quality of service in real-time. Some of the definite contributions are described below:

- We propose a complete infrastructure of IoT along with stream processing techniques, which can facilitate real-time analysis for IoT applications. This is one of the unique attempts in this direction.
- Evaluate the performance for IoT applications to realize the quality of service.
- We perform rigorous simulation along with the integration of DSPS with IoT and analyze the result to get the best possible solution for stream processing.

The rest of the paper is organized as follows: In Section 2, we present the previous work in stream processing technology for IoT. Section 3, introduce a four layer enabling infrastructure for IoT using stream processing. Section 4, presents the introduction for DPSP followed by comparative analysis. In Section 5, we describe two benchmark applications followed by performance evaluation of DSPS for these bench mark applications. Finally, in Section 6, we provide the conclusion of the paper followed by recommendation and future opportunities for IoT.

2. State of the art literature review

IoT is a network of resource-constrained devices to exchange the information in real-time and efficiently along with its vibrant features. Thus, IoT should support a feasible and efficient real-time data processing to its applications. Here, we need a novel stream processing technology to handle real-time data. However, stream processing is widely applicable in various areas, as Cugola et al. [12] presented a detailed overview of stream processing systems along with the concepts and languages that are used to build tools, to handle information flow in an application. Further, it describes the topological style of the system, along with the interaction style between the various tuples. In 2014, Hirzel et al. [13] investigated and compared different stream processing systems and optimization techniques for different stream processing operators. In this, the author presented a catalog of optimizations for stream processing to provide extensive guidance for users.

Initially, stream processing was used in the finance market, while the stock exchange is moving from floor-based trading to current electronic trading. Further, it takes part in distributed computing, parallel computing, and cloud computing. Nowadays, as stream processing is a novel and efficient technology for real-time data processing, evolutionary it becomes the most suitable solution for IoT data processing. Yasumoto et al. [14] presented a survey on the emerging technologies for real-time data processing. In this they clearly state that stream processing is the most imminent and suitable technology for IoT applications. Further, they proposed a three-layer conceptual framework to process and analyze the massive amount of real-time data streams, which is based on the distributed nature of IoT devices.

In 2018, Kazem et al. [15] presented a detailed review of the sustainable system for the smart grid along with the various communication technologies. It provides a detailed comparative solution for various communication technologies to control the smart grid IoT application. In 2019, de Assuncao et al. [16] presented the solution for stream processing technology and techniques to manage resource elasticity for different applications that belong to different research domains. This work is mainly focused on resource elasticity for stream processing engine and elaborate on how one can manage the available resources to process continuous data in real-time. Further, it also elaborated on the programming aspects of stream processing along with the deployment on various applications. In this paper, we discuss various DSPS for IoT applications and evaluate the performance of these systems for IoT applications. In 2019, Wang et al. [17] proposed a Time-based Access Control (TAC) method for multi-attribute data in IoT, which is based on hash chain techniques for resource constraint devices. The proposed technique is adequate for resource-constrained devices, which is an essential enhancement for IoT applications. Further, it considers people-centric IoT applications for massive data collection in real-time and analyzes the data in time series analytical framework. In the same year, Raja et al. [18] proposed a fog computing-based architecture for disaster monitoring to provide qualitative services. Further, it reduces the transmission and computation delay, which is a near real-time solution for disaster management. However, the proposed solution is only applicable to disaster monitoring.

IoT aims to facilitate an intelligent solution in real-time using a variety of sister technologies, which are used to process the data to provide knowledge in the area of interest. With this aim, Catia et al. [19] presents a case study by creating an intelligent environment smart campus, where a member of a particular community can access the related information. It merely visualized the collected data with the help of a web interface from various sensors deployed in the area of interest. Moreover, the end-user can provide control instruction with the help of the application interface.

3. Enabling infrastructure of distributed stream processing system for IoT

Stream processing is a compatible solution for IoT applications, which is integrated with IoT architecture [1]. In the previous sections, we looked into the previous work done in the area of stream processing along with IoT and found that both technologies are beneficial to each other as both fulfill the desired condition for each other. In this section, we have shown how DSPS applied to IoT.

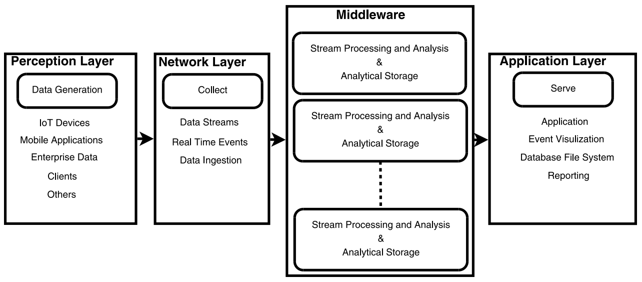


Fig. 1. Flow diagram of IoT for DSPS.

Fig. 1 shows the general data flow of IoT for DSPS along with components, technology, and techniques that process data in IoT applications. It shows all the steps, how the data goes through from the step of data generation to serving the end-users. Each layer has its responsibility to process the data and pipelines from one layer to the next layer. So, it is not necessary to store at any physical location separately, although the system can store analytical results for future consideration.

Fig. 2 presents the overall infrastructure, along with the components involved in DSPS for IoT applications. It consists of four layers. These layers elaborate each stair of data flow from data generation, collection to process till the reporting along with storage and computation, which are as follows:

3.1. Perception layer

As the name suggests, this layer is used to perceive the physical properties of the environment in the area of interest, which is the part of any IoT application. For this, it uses sensing technologies (e.g., TAGs, GPS, Sensor etc.) Further, it converts the physical information into digital signals, which is a suitable form for transmission and communication. Although, several objects might not be perceived, but can perceive with a small microchip. So, the embedded sensor is a good option for sensing purpose here. However, a small microchip attached with sensors facilitates a complimentary benefit to process the data at the end devices. Indeed, embedded technology is a beneficial technology at the perception layer.

The perception layer devices have the quality to sense the environmental data along with the physical interface to the network layer. However, the perception layer devices can choose wireless or wired medium according to the application requirements. Thus, in brief, physical devices use physical addressing to deliver the frame or signal on the network.

To make it ease in understanding, let us assume SN be a set of sensor nodes, where $SN = \{SN_1, SN_2, SN_3, \ldots, SN_N\}$. Thus, the total N number of sensor nodes are used at the perception layer. Moreover, any sensor node x at the physical layer is represented by a set of tuples, which are as follows: $SN_x = \{TimeStamp, MacineID, Location, SD\}$ where, SD represents the set of sensing data, which represent the set of sensing capabilities supported by any sensor node. Further, SD is the combination of sensing type (ST) and sensing value (SV). Thus, $SD = \{ST, SV\}$. Initially, the preprocessed values of the sensor nodes SN is stored in binary format, and physical layer converts that binary data into signals. Further, the perception layer uses the physical address to forward the data to the network layer over the supported (Wired/Wireless) media.

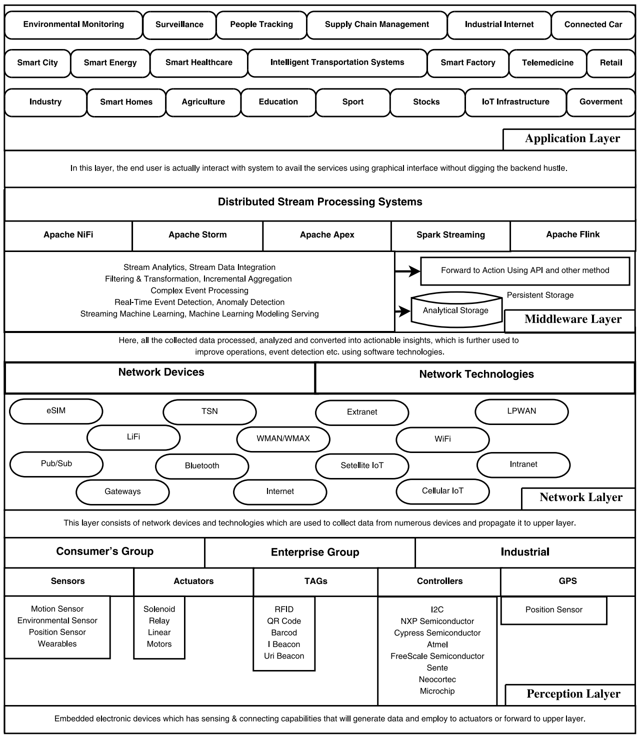

Environmental Monitoring
Surveillance
People Tracking
Supply Chain Management
Industrial Internet
Connected Car
Smart City
Smart Energy
Smart Healthcare
Intelligent Transportation Systems
Smart Factory
Telemedicine
Retail
Industry
Smart Homes
Agriculture
Education
Sport
Stocks
IoT Infrastructure
Goverment
Application Layer
In this layer, the end user is actually interact with system to avail the services using graphical interface without digging the backend hustle.
Distributed Stream Processing Systems
Apache NiFi
Apache Storm
Apache Apex
Spark Streaming
Apache Flink
Stream Analytics, Stream Data Integration
Filtering & Transformation, Incremental Aggregation
Complex Event Processing
Real-Time Event Detection, Anomaly Detection
Streaming Machine Learning, Machine Learning Modeling Serving
Forward to Action Using API and other method
Persistent Storage
Analytical Storage
Middleware Layer
Here, all the collected data processed, analyzed and converted into actionable insights, which is further used to improve operations, event detection etc. using software technologies.
Network Devices
Network Technologies
eSIM
TSN
Extranet
LPWAN
LiFi
WMAN/WMAX
WiFi
Pub/Sub
Bluetooth
Setellite IoT
Intranet
Gateways
Internet
Cellular IoT
Network Lalyer
This layer consists of network devices and technologies which are used to collect data from numerous devices and propagate it to upper layer.
Consumer's Group
Enterprise Group
Industrial
Sensors
Actuators
TAGs
Controllers
GPS
Motion Sensor
Environmental Sensor
Position Sensor
Wearables
Solenoid
Relay
Linear
Motors
RFID
QR Code
Barcod
I Beacon
Uri Beacon
I2C
NXP Semiconductor
Cypress Semiconductor
Atmel
FreeScale Semiconductor
Sente
Neocortec
Microchip
Position Sensor
Perception Lalyer
Embedded electronic devices which has sensing & connecting capabilities that will generate data and employ to actuators or forward to upper layer.

Fig. 2. Enabling infrastructure of IoT for stream processing.

3.2. Network layer

This layer is liable to collect the data from the perception layer. Further, it transmits the data to the application layer with the help of various network technologies (e.g., LAN, MAN, LPWAN, Bluetooth, LiFi, etc.) In most of the IoT systems, wireless technologies (e.g., 3G/4G/5G, WiFi, Bluetooth, Zigbee, etc.) are the most useful technologies to carry the data over the network. In most of the IoT applications, data is massive in quantity. So, it is complicated to store this massive amount of data and process it to make a decision.

Here, the need for stream processing technologies comes into the picture to process the data in motion. However, this layer is only responsible for handling the vast amount of data transmission traffic over the network. For this, it uses network devices (e.g., Gateways, Routers, etc.) to handle the continuous data and perform data preprocessing at the network layer (if required depends on the application). Further, this layer uses various network technologies (e.g., Internet, WiFi, etc.) to forward the data at the middleware layer. Typically, it uses Internet Protocol (IP) stack to transmit the data packet using logical addressing schema.

3.3. Middleware layer

This layer is responsible for providing a software interface to handle the data from the perception layer and provide concentrated results to the application layer. It provides the required abstraction to hide the complexity of used technologies in the whole system. Indeed, it is a necessary layer to divide the end-users and developers, which works on the complex processing technologies used by the lower layers. It allows developers to deal with various designing issues. Hence, it bifurcates the developers in investing the time to think about the utilization of the physical layer technologies. It is merely responsible for handling the data by processing, analyzing, and intimating about the events in the area of interest.

This layer acts as a software bridge between smart objects and the applications. Further, an Application Programming Interface (API) (e.g., JavaScript Object Notation (JSON), Simple Object Access Protocol (SOAP), Representational State Transfer (REST), etc.) between abstract hardware and application is used for data management, security, privacy, and computation. Moreover, the primary purposes of the middleware are to provide seamless interaction among the various aspects of the application or between the applications along with efficient connectivity, concise translation period, innovative solution, and easily accessible workplace tools for applications.

3.4. Application layer

In the case of IoT, there is a wide variety of areas in which intelligent processing has been developed. Although all the applications are not available to use; however, the primary finding indicates the scope of improvement towards the quality of life.

This layer is responsible for using the processed data in the form of knowledge, which is served as a result of the end-users. Most of the time, the served knowledge is in the form of graphics using a graphical interface (e.g., websites, applications, etc.). Moreover, this layer facilitates some required tools (e.g., actuating devices) to realize the IoT vision. Thus, the application layer is the only interface for end-users. Moreover, the end-user can provide instruction from this application interface to provide the instructions according to its requirements.

4. DSPS for IoT

In this section, we describe the five most popular, open-source, and best-distributed stream processing frameworks, which we are going to utilize for IoT applications.

Apache software foundation provides powerful and robust data processing frameworks, which support large scale data processing. Moreover, the Apache software foundation is a decentralized and non-profit corporation, which was introduced as an open-source community in front of the developers from all over the world. Further, Apache provides a set of open-source frameworks that simplify the development of stream processing applications. Moreover, to the best of our domain knowledge, Apache provides the best continuous (streaming data) reliable and scalable framework for real-time applications. Thus, we have used the frameworks provided by Apache as benchmark platforms for IoT applications.

We have selected these systems according to their feature and compatibility with IoT applications and found that these five systems are the most permissible solution for IoT applications, which are defined below:

4.1. Apache NiFi [20]

It is an open-source software solution developed by the apache software foundation, which takes care of the automation of distributed data flow among the systems. The name NiFi inherited from the "Niagara Files" software, developed by National Security Agency (NSA).

The core design concept of NiFi is based on a flow-based programming model. Moreover, it offers the features to operate within the clusters and security using transport layer security encryption along with the extensibility features using various plugins. So, the user can add features to enhance the ability of the software. Fig. 3 shows the architecture of NiFi. It executes on the host operating system in Java Virtual Machine (JVM), which has a set of components described as:

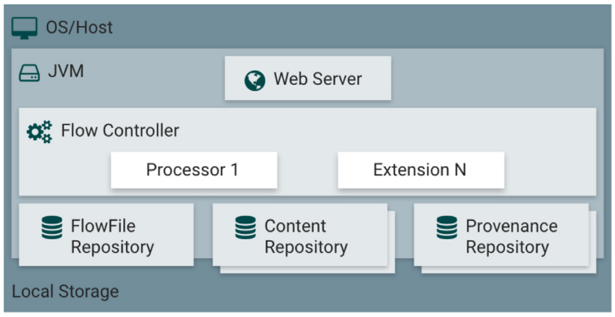


Fig. 3. General architecture of Apache NiFi.

Web server is responsible for all the communications with NiFi using Hyper Text Transfer Protocol (HTTP) and Application Programming Interface (API). Second is the flow controller, which is the core part of NiFi to perform operations. It also provides extension threads to receive extensive resources to execute them. Further, there are three types of repositories. FlowFile Repository keeps tracks of the presently active module in the flow. Content Repository is responsible for live the content bytes from Flow-File. Finally, Provenance Repository stores the event data in the physical database. The repositories used are pluggable according to the user requirements, which make it a scalable solution of IoT applications. Further, it can also operate within the cluster and the ZooKeeper server, as a coordinator is used to keep the synchronization among multiple nodes.

4.2. *Apache Storm [21]*

The storm is a highly distributed and widely adopted real-time computational system, initially introduced by Nathan Marz and the team. Further, this project was open sourced since it was acquired by Twitter [14]. However, its working is similar to Hadoop batch processing up to some level. Although, storm works on unbounded data streams reliably. The high-level architecture of the storm is shown in Fig. 4.

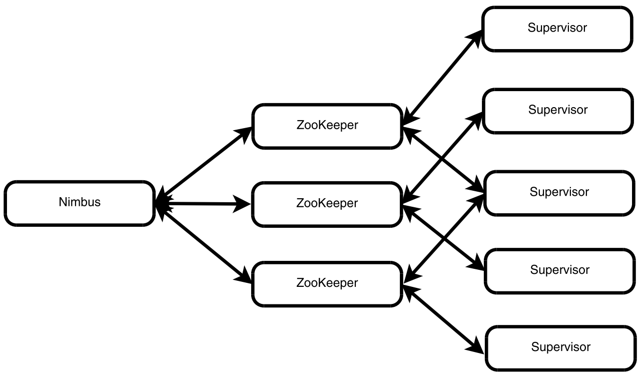


Fig. 4. General architecture of Apache Storm.

Its clusters consist of two types of nodes i.e., master node and worker node. Initially, the master node runs a daemon called Nimbus, which takes care of tasks assigned to worker nodes and monitors the performance of the worker nodes. Worker nodes take the data stream and execute them. Further, the worker node executes a daemon called supervisor, which is handled by Nimbus and starts and stops the process according to the instruction. Since storm clusters are unable to manage its state, so for this purpose it relies on Zookeeper. Further, Zookeeper facilitates communication between Nimbus and supervisors using message exchange.

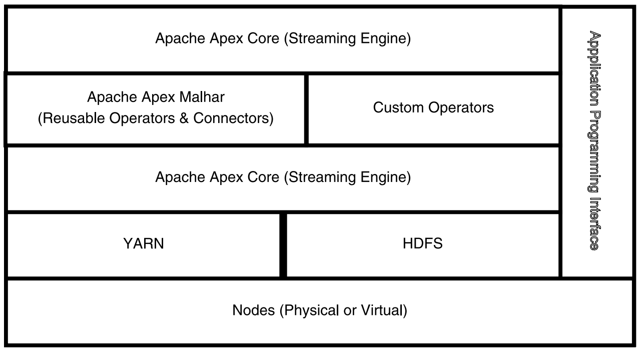

Apache Apex Core (Streaming Engine)
Apache Apex Malhar (Reusable Operators & Connectors)
Custom Operators
Apache Apex Core (Streaming Engine)
YARN
HDFS
Application Programming Interface
Nodes (Physical or Virtual)

Fig. 5. General architecture of Apache Apex.

4.3. *Apache Apex [22]*

It is a YARN-native platform that combines stream and batch processing. Originally developed under the project directed by San Jose, California based startup Data Torrent. Apex is a highly scalable, distributed, fault tolerant, secure, stateful, easily operable, and high performance system, which processes big data in motion. It consists of two parts Apex Core and Apex Malhar, as shown in Fig. 5.

Apex Core is the central part of the Apex that provides the framework for distributed applications over Hadoop for stream processing [23]. Apex Malhar provides the required operators and connectors to enable rapid application development. These operators are responsible for providing an interface with source, sink, and other database connectors. Apex can work with both stream processing and batch processing at the same time, which provides a real-time solution.

4.4. *Spark Streaming [24]*

Spark is a fast and general purpose computing system, which works within clusters. It is a highly scalable and widely adapted stream processing engine for data stream processing, and it is also work in batch mode that makes it highly scalable and high performance with a high volume of data. It supports in-memory processing to achieve high performance for scalable applications. However, it can also perform batch processing when application produces a massive amount of data that is available into memory space.

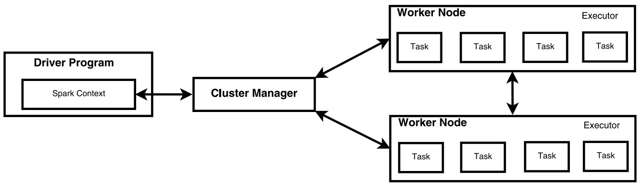


Fig. 6. General architecture of Spark Streaming.

Fig. 6 shows the overall architecture of Spark. It consists of three main components: Driver, Cluster Manager, and Executers. Spark Driver is the master node, which takes care of overall data stream processing and reports the processed data. Spark Cluster divides the application into a set of tasks, which would be executed by a set of executors. Once the driver divides the application into a set of task, and forwards to the cluster manager, which is responsible to distribute the tasks into executors according to their features. Each executor is responsible for executing the assigned task and reporting to the cluster manager. Further, the cluster manager combines the collected result and forwards it to the driver.

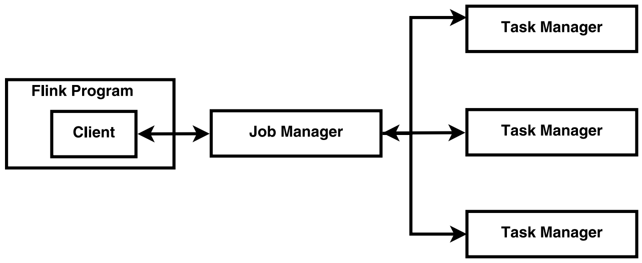


Fig. 7. General architecture of Apache Flink.

4.5. *Apache Flink [25]*

Flink is a distributed processing system, which provides stateful computation for real-time data streams. The central component of the flink is a dataflow engine written in Java and Scala, which executes the data in a parallel and pipelined manner. It also supports batch processing along with iterative algorithms. It can execute the customized user-defined function through system stack.

Fig. 7 shows the underlying architecture of Flink, which is based on master–slave architecture. It consists of job manager and multiple task managers. A job manager is responsible for distributing the tasks among the task managers for parallel processing. Further, task managers are used as worker nodes and execute their part parallelly. Further, the user can import the customized program using the client with the help of API.

We have analyzed several open-source stream processing systems to process events in the area of interest in real-time, which is an efficient solution for IoT applications. We have chosen five of these platforms of various categories as General Processing (GP), Event Stream Processing (ESP), and Complex Event Processing (CEP), which supports the deployment of stream processing along with the development.

In this direction, the above five technologies are the leading system, which fulfills the desired features required for the current scenario of Internet of People paradigm. Table 1 provides a comparative overview of stream processing systems. Any person or organization developing the IoT application should consider the stream processing technologies as they choose the stack and architecture to their system by considering the people as a part of system. The above technologies offers a new and important way to look at the data, which facilitate a real-time and scalable solution. Thus, we found these softwares as the most suitable solution for IoT applications as well as for Internet of People paradigm. Moreover, we compare the above middleware softwares with other existing middleware software.

4.6. *Comparison of Apache frameworks with existing similar frameworks*

In this section, we have shown a comparison with existing similar middleware over some essential features. Several software vendors provide middleware support for a wide verity of streaming processing applications. The processing uses the middleware to perform various computation operations, like pattern matching, averaging, event filtering, event prediction, and find the correlation among the events over the input data streams or dataset. The middleware takes the mined data as input according to custom application and publishes the results accordingly. Apache NiFi, Apache Storm, Apache Storm, Apache Apex, Spark Streaming, Apache Flink, RabbitMQ, TIBCO StreamBase, IBM Infosphere Streams, Amazon Kinesis, Fujitsu Interstage CEP, etc. are the famous and notable example middleware platforms. In Table 2, we compare the Apache middleware with other open and closed source middleware over some essential features for data stream processing.

Table 2 provides a high-level view of why we only choose Apache middlewares in our work. In Table 2, We have shown a necessary summary of middlewares by considering several essential features for comparison. Further, a detailed version of this comparison, which elaborates on other existing middleware, can be found in [31]. Further, we evaluate the performance of these system for two different applications, which are described in next section.

5. Performance evaluation

In this section, we evaluate the performance of the above systems for IoT applications. Further, we performed various simulations in order to appraise the performance of the stream processing systems for IoT effectively. Our main motive is to get high throughput and low latency, as these are the main requirements for real-time data processing. Further, we have presented the process flow diagram to make it easy to understand the flow for performance evaluation.

5.1. *Process flow diagram for applications*

Process Flow Diagram (PFD) has illustrated the data processing using stream processing for IoT applications. Further, Fig. 8 provides a complete picture to understand data processing using stream processing.

Node-RED provides the front end, which is used to access the data from different sources using different techniques like RESTful application programming interface, MQTT data feeds, etc. Further, it can apply preprocessing techniques like filtering, data formatting, and forward the data streams to the stream processing engine using a dedicated procedure of stream processing engines (like in storm processing framework broker is used to handle the data). Further, the stream processing engine process the data and make it available to store in the dedicated storage unit (like Data Lake provided by different commercial organizations) and web and other application interfaces to publish the data.

We define the dataset of two important IoT applications, which we have used in the performance measurement of the above stream processing systems followed by the performance evaluation.

5.2. *Experimental data*

To evaluate each DSPS system, we create near real-time scenarios for the dataset of two different applications one is Fire Information for Resource Management Systems (FIRMS) [32] and other is Medical Information Mart for Intensive Care (MIMIC) [33], which are two novel and widely available datasets for IoT applications to simulate real-time stream processing system.

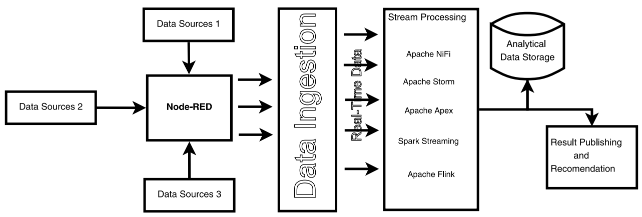


Fig. 8. General process flow diagram for IoT applications.

Table 1
Comparative analysis of different DSPS.

Features	Apache NiFi [20]	Apache Storm [21]	Apache Apex [22]	Spark Streaming [24]	Apache Flink [25]
Processing system	ESP and CEP	GP	GP and CEP	GP and ESP	GP and CEP
Programming model	Flow-based	Imperative	Imperative and declarative	Imperative and declarative	Imperative
Infrastructure	Cluster, Fog	Cluster, Fog, Cloud	Cloud, Fog	Cluster, Fog, Cloud	Cluster, Fog, Cloud
Parallelization	Data and task	Data and task	Data	Data and task	Data and task
Processing	Stream	Batch and stream	Batch and stream	Stream	Batch and stream
Stateful	Yes	Yes	Yes	Yes	Yes
Real-time	Yes	Near real-time	Near real-time	Yes	Yes
API	Compositional	Compositional	Declarative	Declarative	Declarative
API language	REST	Scala, Java, Clojure, Python, Ruby	Java	Scala, Java, Python	Java, Scala, Python
Data flow	Flow-based	DAG	Pipelined	DAG	Dependency Tree
Scalability	High	Medium	Medium	High	High
Fault tolerant	Yes (locally)	Yes (worker is responsible)	Yes (YARN takes care of it)	Yes (with no extra code)	Yes (based on lightweight distributed snapshots)
Security	Password authentication	User authentication via Kerberos	User authentication via Hadoop	Password authentication	User authentication via Hadoop/kerbros

Table 2
Comparison of Apache frameworks with existing similar frameworks.

Features	RabbitMQ [26]	TIBCO StreamBase [27]	IBM Infosphere Streams [28]	Amazon Kinesis [29]	Fujitsu Interstage CEP [30]	Apache NiFi [20]	Apache Storm [21]	Apache Apex [22]	Spark Streaming [24]	Apache Flink [25]
Distributed processing	FS	FS	FS	FS	NS	FS	FS	FS	FS	FS
Fault tolerance	LS	NS	FS	FS	NS	FS	PS	PS	FS	FS
Scalability	LS	FS	FS	LS	NS	FS	PS	PS	FS	FS
API Support and extension for information processing	PS	NS	NS	FS	FS	FS	FS	FS	FS	FS
User interface	PS	FS	FS	FS	PS	FS	FS	FS	FS	FS
Debugging	PS	FS	FS	FS	NS	FS	FS	PS	FS	FS
Best reported performance	NA	91,000 evals/s	7.3 Million tuples/s	NA	Up to 2 Million events/s	NA	3.2 Million tuples/s	3.3 Million tuples/s	130,000 events/s	1.5 Million events/s
Open source license	YES	NO	NO	NO	NO	YES	YES	YES	YES	YES

FS:- fully supported, PS:-partial supported, NS:-not supported, LS:-least supported.

This data is generated by different sensors based on physical characteristics. As a result, the generated data points are in the forms of time-series as well as spatial, which are generated periodically by the sensors. However, the sampling rate for used sensors may vary from once a day to hundreds of data points per second, according to the applications domain. Moreover, the number of sensors used may also vary according to the applications or geographical locations.

IoT applications, like smart health monitoring, can generate a large number of data points at the rate of thousands of messages per second, while in case of forest monitoring, the generated data point may vary up to a hundred messages per second. Therefore, we can visualize a wide range of input throughputs, which are the ideal and compatible situation to simulate real-time IoT applications.

5.2.1. Fire Information for Resource Management Systems (FIRMS) [32]

FIRMS is a global forest monitoring system, which derives hotspots and fire locations. Moreover, this system uses a wide variety of sensors, which is used to detect the fire across the world. We have used the fires for Indonesia; this dataset consists of active fire data for Indonesia from 1 January 2013 to 12 December 2019 which is updated daily. This dataset consists of 460,556 Records.

5.2.2. Medical Information Mart for Intensive Care (MIMIC) [33]

MIMIC is the widely and freely available health related database of over forty thousand patients who were admitted in critical care units of the Beth Israel Deaconess Medical Center from 2001 to 2012. This dataset consists of information related to patients such as demographics, laboratory test reports, sign measurements, imaging reports, caregiver notes, medications, and mortality. Moreover, it consists of high temporal data points related to lab results, digital information to monitor trends. We use this database because it contains diverse and colossal population data who admitted to ICU. Moreover, it is freely available to the research community worldwide.

These datasets conform to the real values collected from the related domains. In order to scale these data streams, we make temporal and spatial scaling. Temporal scaling allows scaling down the data in a time-series format that were generated over

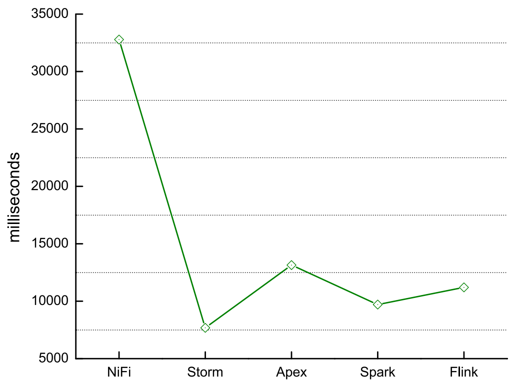


Fig. 9. Time taken by the DSPS to start for a data stream.

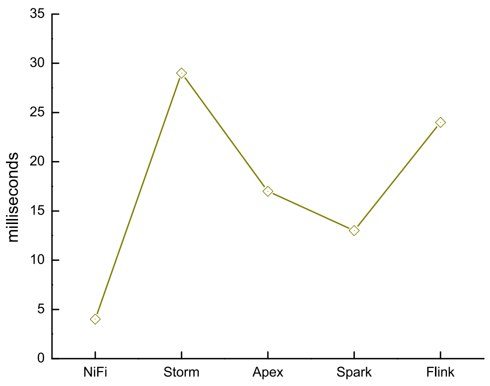


Fig. 10. Average response time for DSPS.

a period of time. For example, MIMIC data is scaled into multiple of 50, as compared to its original data rate.

On the other hand, we can use spatial scaling when more significant numbers of sensors are used. This is necessary in case of a small sample of sensors. So, we consider data streams of the same sensor for the different windows. These two schemes are used to normalize the datasets. Moreover, these datasets represent the real-world scenarios, which achieve the diversity in distribution rate for input streams and facilitates the rate of the span from 100 000 to 20 000 000 messages/s.

5.3. Experimental setup

We evaluate the above systems for the IoT applications data. In order to that, we deployed these systems on a local high-performance computer having 64 Core, which consists of multiple virtual machines of 2GB RAM each. We use the Linux Ubuntu 19.04 operating system to create the environment. We use multiple clusters and calculate the average output in order to evaluate the performance of the systems.

5.4. Evaluated results

In this section, we evaluate performance in terms of startup time, response time, throughput, jitter, and scalability in-terms of throughput for the above stream processing systems, as follows:

5.4.1. Startup time

It refers to the time consumed by DSPS to be ready to respond for input. In our evaluation, NiFi takes the longest time for startup and Storm takes the shortest time for startup, which is shown in Fig. 9.

5.4.2. Response time

It is defined as the total time elapsed between the service requested and the time when the system responds to that service [34]. In other words, we can define it as the total time the system takes from when a user requests until the system is ready to serve the request. Response time is affected by both processing time and the latency, which is depending on the hardware resource or utilization. Processing time refers to the amount of time that the system takes to process a request and latency refers to the delay in communications.

In a real-time environment, it is an essential factor and should be low or almost negligible. For example, maximum response time for civil aircraft is 50 ms and in the case of military aircraft, it is 5 ms depends on the criticality of the applications. Thus, the response time should be as low as possible. Further, various factors are affecting the latency like resources, size of the data streams and input rate.

In general response time RT is represented as:

$$RT_i = PT_i + I_i \tag{1}$$

where, PT_i:- processing time of a task
I_i:- maximum amount of delay during execution

Moreover, if there is any previous task executed on the system then, response time depends on the previous task, which can be defined as:

$$RT_i(t) = PT_i + \sum_{j=1}^{i-1} \left\{ \frac{t}{P_j} \right\} PT_j \tag{2}$$

where, PT_i:- processing time
P_j:- period of previously scheduled task
PT_j:- processing time of previously scheduled task. In our work, we visualize the average response time of each DSPS for both applications, as shown in Fig. 10. We can observe that NiFi provides the minimum average response time for the tested data streams, which fulfills the most necessary requirement of the real-time processing. Thus, NiFi is the most suitable solution for real-time applications.

5.4.3. Throughput

It measures the processing capability of the system, as to how much data streams of information a system can process in a unit amount of time. Although, latency and throughput are not correlated. In the case of stream processing, data streams together (to form a batch data) can achieve higher throughput. Although, time spend to form that batch can increase the latency in the system.

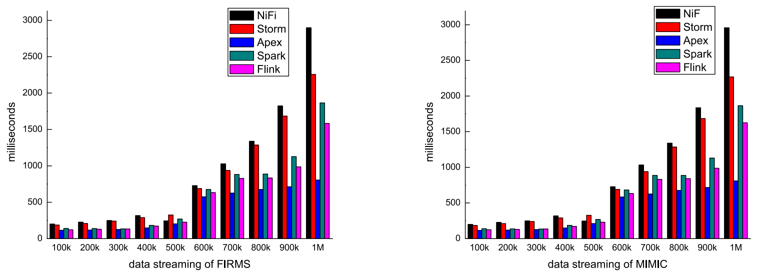
Practically, the deployment of DSPS considers the arrival rate of data streams and has to scale according to the arrival rate. As the arrival rate of information increases the DSPS needs to handle it by scale out the data streams to make uninterpreted processing fluently.

In this paper, we calculate throughput according to the input rate and selectivity from that. Thus, TH^o is calculated using:

$$TH^o = \phi \times TH^i \tag{3}$$

where:
ϕ is the selected input rate
TH^o is the output throughput
TH^i Input data stream rate

Further, the selected input rate is based on the stream processing system capabilities, which varies according to system. Fig. 11 shows the throughput of stream processing for both of the



(a) Forest Monitoring in indonesia(b) Health Monitoring in Beth Israel

Fig. 11. Throughput for different applications.

applications. Further, we observe that NiFi provide the highest throughput for both applications, which is approximately more than 2 times of Apex. Thus, NiFi is the most suitable solution for IoT applications.

5.4.4. Jitter

It may be possible that the resultant throughput may deviate from the ideal output because of variations in the input streams of data or change in the path for the dataflow. Precisely, in our context, it is the difference between the earliest and the latest time a job could be ready to use relative to invocation (arrival) of the assigned job and its release.

Although there are many reasons for the occurrence of jitter like, the system uses parallel processing or some hazardous effect while implementing the assigned task or job. In our work, jitter is the variation between resultant throughput and expected throughput over a time interval.

In this paper, we have used jitter to track the variation in the output throughput as compared to the expected output over a time interval t, as follows:

$$JT_t = \frac{TH^o - \phi \times TH^i}{\phi \times TH^{ai}} \tag{4}$$

where:
ϕ is the selected input rate
TH^o is the output throughput
TH^i Input data stream rate
TH^{ai} long term average input rate

Jitter tends to zero for an efficient system and it should not be affected by the variation in the input streaming rate. So, it should be constant or nearer to zero for an efficient stream processing system. We have calculated the jitter for both applications irrespective of average relative throughput, as shown in Fig. 12. Moreover, we observe that NiFi provides the smallest possible jitter near about zero and Apex gives the maximum possible jitter.

5.4.5. Scalability analysis

It defines the capacity of the system or handling capacity of a system when the amount of workload increases. In other words, it defines the accommodation of enlargement of the system.

In case of IoT applications, an enormous amount of data floating through the system in real-time. Moreover, the sensed data is moderate, which needs to process without fail. The computing capacity of the system is responsible for handling this data using the scaling method. Here, we need to know how our system will behave with scaling methods. In this direction, we have to find a way to analyze the system, which can provide the system behavior towards the scaling of the system.

We use Universal Scalability Law (USL), which initially proposed by N Gunther [35]. Further, it is based on a black-box approach for modeling, which considers interprocess communication at all levels of the proposed framework (i.e., application level, middleware level, network level, and hardware level).

N Gunther [35] defines the scale up capacity $C(n)$ of the system for n number of nodes with following equation, as follows:

$$C(n) = \frac{n}{1 + \alpha(n-1) + \beta n(n-1)} \tag{5}$$

where:
n represent the number of distributed nodes
α represent contention, which considers the serial fraction of the input workload
β represent coherency, which considers the delay for interprocess communication

Further, we know that scale up capacity of a system is defined as the ratio of absolute throughput with n nodes and throughput with 1 node.

$$C(n) = \frac{TH^n}{TH^1} \tag{6}$$

Thus,

$$TH^n = C(n) \times TH^1 \tag{7}$$

Here, TH^n represents the absolute throughput, and $C(n)$ represents the relative capacity of the system. However, we need to estimate the value of TH^1 using interblock estimation of the existing available data. Using Eqs. (5) and (7).

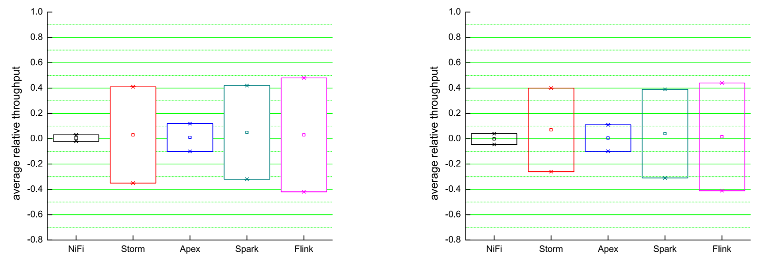
$$TH^n = \frac{n \times TH^1}{1 + \alpha(n-1) + \beta n(n-1)} \tag{8}$$

Here, TH^n represent the absolute throughput with n distributed nodes. Moreover, USL makes it easy and efficient to get absolute throughput with the estimation of TH^1, which incorporated into the same statistical procedure that is used to estimate α and β parameters performed on the previous and currently available data.

We have evaluated the scalability of the DSPS's using a variety of distributed processing nodes to process the data with a dedicated input rate of 500k tuple/s for FIRMS and 1M tuple/s for MIMIC. We use these input rates to maximize the utilization of processing nodes and find how much throughput individual frameworks provide at the dedicated input load.

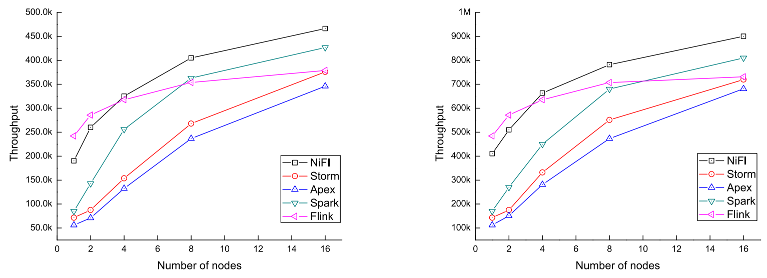
Fig. 13 shows the scalability of the DSPS's in terms of throughput for both of the applications. Further, we have observed that Flink had the lowest scalable solution, and NiFi graph curve shows the continuous improvement in the scalability.

We can see from the results that the startup time of NiFi is high as compare to other DSPS, but NiFi provides high throughput and low response time along with the minimum jitter, which makes it the most suitable solution for IoT applications. Moreover,



(a) Forest Monitoring in indonesia (b) Health Monitoring in Beth Israel

Fig. 12. Jitter for different applications.



(a) Forest Monitoring in indonesia (b) Health Monitoring in Beth Israel

Fig. 13. Scalability analysis for different applications.

Apex is the least useful solution for IoT application as it provides low throughput, high response time, and maximum jitter, which show the maximum variation in throughput. Further, for scalability, Flink provides minimum moderation in the scalability graph, which makes it the least scalable solution of real-time applications. Overall, NiFi provides the best performance for both of the applications. Thus, we find NiFi as the most suitable solution for real-time applications.

6. Conclusion and future opportunities

Nowadays, IoT is the widely accepted technology for most of the organizations as well as for research community people, who facilitate an efficient automated and real-time solution to a wide variety of IoT applications. In this work, we propose a four-layer enabling infrastructure using DSPS for IoT applications. Moreover, we have evaluated the performance of the five most popular and suitable DSPS. To the best of our knowledge, these five middleware supports all the essential features of IoT applications. In support of our argument, we have presented a comparison with other existing technologies in Table 2. Thus, we conclude that Apache facilitates the best and widely accepted solution for stream processing. Further, we presented a process flow diagram to understand the implementation of DSPS for IoT applications.

We have chosen two benchmark applications, one for forest monitoring and another for health monitoring, because these two applications are broadly accepted by the research community. Further, we have evaluated the performance of all DSPS against IoT applications over various performance metrics. On the basis of the evaluated results, we suggest that NiFi is the most suitable and adapted solution for IoT applications. NiFi supports most of the features required for stream processing. Moreover, it facilitates the extension support to extend it to another dimension of stream processing. Thus, it shows outperformance NiFi over other DSPS.

In Section 5.4, we can see that NiFi takes the longest time to start and has the lowest response time compared to the other four DSPS. Further, it clearly illustrates that NiFi outperforms Flink approximately two times, in term of throughput along with the lowest possible jitter. The reason for this is, it provides fast data processing with low latency. Moreover, initially, NiFi provides low scalability in comparison to Flink, but as the size increases, NiFi performs better than the rest of DSPS. Thus, we conclude that NiFi is a highly acceptable solution compared to other existing DSPS for overall performance matrices.

In this paper, our main purpose is to provide a suitable and scalable DSPS with minimum latency and high throughput for IoT applications. However, we have not covered security analysis in our work, as it is a broad and separate research domain for IoT applications. However, we are working towards the security model for IoT applications. We are also exploring the various aspects of the applicability of artificial intelligence in various real-time or time-critical applications. Thus, we would recommend exploring the various aspects of security and prescriptive analytical models in future directions.

CRediT authorship contribution statement

Vikash: Conceptualization, Methodology, Software, Formal analysis, Investigation, Resources, Writing - original draft, Writing - review & editing, Visualization. **Lalita Mishra:** Conceptualization, Formal analysis, Investigation, Resources, Validation. **Shirshu Varma:** Conceptualization, Validation, Supervision.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

References

[1] F. Xhafa, B. Kilic, P. Krause, Evaluation of IoT stream processing at edge computing layer for semantic data enrichment, Future Gener. Comput. Syst. 105 (2020) 730–736.
[2] H. Röger, R. Mayer, A comprehensive survey on parallelization and elasticity in stream processing, ACM Comput. Surv. 52 (2) (2019) 36.
[3] I.R. 7826, Real-Time Streaming Protocol Version 2.0, IETF, 2016.
[4] A. Zoitl, V. Vyatkin, Iec 61499 architecture for distributed automation: The glass half full view, IEEE Ind. Electron. Mag. 3 (4) (2009) 7–23.
[5] H. Kilov, P.F. Linington, J.R. Romero, A. Tanaka, A. Vallecillo, The reference model of open distributed processing: Foundations, experience and applications, Comput. Stand. Interfaces 35 (3) (2013) 247–256.
[6] G. Disterer, Iso/iec 27000, 27001 and 27002 for information security management, J. Inf. Secur. 4 (2013) 92–100.
[7] I. ISO 2394:2015, General Principles on Reliability for Structures, ISO, Zurich, 2015.
[8] I. 18882:2017, Information Technology – Telecommunications and Information Exchange Between Systems – Ubiquitous Green Community Control Network: Heterogeneous Networks Convergence and Scalability, ISO, Zurich, 2017.
[9] I..O. Report, Functional Safety of Electrical/Electronic/Programmable Electronic Safety-Related Systems, IEC, 2006.
[10] S. Dickinson, T. Lemon, T. Pusateri, DNS Stateful Operations, IETF RFC 8490, 2019.
[11] J.N. Hasoon, A.P.D.R. Hassan, Big data techniques: A survey, Iraqi J. Inf. Technol. 9 (4) (2019) 135–146.
[12] G. Cugola, A. Margara, Processing flows of information: From data stream to complex event processing, ACM Comput. Surv. 44 (3) (2012) 15.
[13] M. Hirzel, R. Soulé, S. Schneider, B. Gedik, R. Grimm, A catalog of stream processing optimizations, ACM Comput. Surv. 46 (4) (2014) 46.
[14] K. Yasumoto, H. Yamaguchi, H. Shigeno, Survey of real-time processing technologies of iot data streams, J. Inf. Process. 24 (2) (2016) 195–202.
[15] K. Sohraby, D. Minoli, B. Occhiogrosso, W. Wang, A review of wireless and satellite-based m2m/iot services in support of smart grids, Mob. Netw. Appl. 23 (4) (2018) 881–895.
[16] M.D. de Assuncao, A. da Silva Veith, R. Buyya, Distributed data stream processing and edge computing: A survey on resource elasticity and future directions, J. Netw. Comput. Appl. 103 (2018) 1–17.
[17] B. Wang, W. Li, N.N. Xiong, Time-based access control for multi-attribute data in internet of things, Mob. Netw. Appl. (2019) 1–11.
[18] G. Raja, A. Thomas, Safer: Crowdsourcing based disaster monitoring system using software defined fog computing, Mob. Netw. Appl. (2019) 1–11.
[19] C. Prandi, L. Monti, C. Ceccarini, P. Salomoni, Smart campus: Fostering the community awareness through an intelligent environment, Mob. Netw. Appl. (2019) 1–8.
[20] Apache NiFi, 2019, https://nifi.apache.org/ [accessed 15 December 2019].
[21] Apache Storm, 2019, http://storm.apache.org/ [accessed 15 December 2019].
[22] Apache Apex, 2019, https://apex.apache.org/ [accessed 15 December 2019].
[23] Apache Hadoop, 2019, https://hadoop.apache.org/ [accessed 15 December 2019].
[24] Spark Streaming, 2019, https://spark.apache.org/ [accessed 15 December 2019].
[25] Apache Flink, 2019, https://flink.apache.org/ [accessed 15 December 2019].
[26] T. Sharvari, N.K. Sowmya, A study on modern messaging systems-kafka, RabbitMQ and NATS streaming, arXiv(2019)arXiv–1912.
[27] R. Tibbetts, Performance & Scalability Characterization, StreamBase, 2009.
[28] H. May, D. Engebretsen, W. Madden, IBM infosphere streams v4.0 performance best practices, 2019, https://developer.ibm.com/streamsdev/wp-content/uploads/sites/15/2015/04/Streams_4.0.0.0_PerformanceBestPractices.pdf [accessed 15 December 2019].
[29] S. Gulabani, Practical Amazon EC2, SQS, Kinesis, and S3: A Hands-On Approach to AWS, Apress, 2017.
[30] FUJITSU, Interstage business process manager v11 architecture whitepaper, 2019, https://www.fujitsu.com/global/documents/products/software/middleware/application-infrastructure/interstage/download/bpm/Fujitsu-Interstage-BPM-V11-Architecture-Whitepaper.pdf [accessed 15 December 2019].
[31] M. Dayarathna, S. Perera, Recent advancements in event processing, ACM Comput. Surv. 51 (2) (2018) 1–36.
[32] Fire information for resource management systems (FIRMS), 2019, http://data.globalforestwatch.org/ [accessed 15 December 2019].
[33] A.E. Johnson, T.J. Pollard, L. Shen, H.L. Li-wei, M. Feng, M. Ghassemi, B. Moody, P. Szolovits, L.A. Celi, R.G. Mark, Mimic-iii, a freely accessible critical care database, Sci. Data 3 (2016) 160035.
[34] C. Avasalcai, S. Dustdar, Latency-aware decentralized resource management for IoT applications, in: Proceedings of the 8th International Conference on the Internet of Things, 2018, pp. 1–4.
[35] B. Schwarz, Practical Scalability Analysis with the Universal Scalability Law, VividCortex, Charlottesville, VA, USA, 2015.

Vikash received the B.Tech. Degree in Computer Science & Engineering from Uttar Pradesh Technical University, Lucknow, India, in 2013, and M.Tech. Degree in Computer Science & Engineering from Kamla Nehru Institute of Technology, Sultanpur, India, in 2015. After that, He was Assistant Professor in the department of Information Technology, Greater Noida Institute of Technology, Greater Noida, India. Currently, he is pursuing Ph.D. at Indian Institute of Information Technology, Allahabad in the department of Information Technology. His current research interests include Internet of Things, Wireless Sensor Networks, Pervasive Computing, Middleware, Wireless and Mobile Computing.

Lalita Mishra received the B.Tech. degree in Computer Science & Engineering from Uttar Pradesh Technical University, Lucknow, India, in 2011 and M. Tech. from International Institute of Information Technology, Bhubaneswar. Currently, she is working towards Ph.D at Indian Institute of Information Technology, Allahabad in the department of Information Technology.

Her research interest include Internet of Things, Cloud Computing, Wireless Sensor Networks, Edge Computing.

Shirshu Varma after completing Ph.D. served many reputed organizations like, BIT Mesra Ranchi, CDAC Noida in the capacity of lecturer, Sr. lecturer & Principal project engineer. Presently working as a Professor in Indian Institute of Information Technology-Allahabad (IIIT-Allahabad).

He has about 25 years of experience of teaching and research. He has published about 50 papers in international and national journals and conferences of repute and is the author of 04 book chapters. His area of work includes-: Wireless sensor networks — coverage and connectivity, Sensor deployment and localization, Wireless sensor statistical routing etc., WI-FI, WiMAX.