Multimedia Tools and Applications (2024) 83:37369–37385
https://doi.org/10.1007/s11042-023-17151-6

# An efficient architecture for processing real-time traffic data streams using apache flink

B. Gnana Deepthi[1] · K. Sandhya Rani[1] · P. Venkata Krishna[1] · V. Saritha[1]

Received: 7 September 2022 / Revised: 1 September 2023 / Accepted: 15 September 2023 /
Published online: 30 September 2023

**Abstract**

Big Data technologies emerging day by day and are making drastic changes in various real-world applications. Traditional data mining tools adequate to process volumes of data but from past decades the rapid growth in data becomes difficult for processing. Due to continuous flow of data, data streams require additional computational processing than the traditional one. Big data stream processing considers different features of the data streams heterogeneity, scalability, fault tolerance and query optimization. Efficient implementation of these features in real-world applications using big data analytics is a challenging job during data storage, processing, and analysis phases. Therefore, the proposed model FRTSPS is a generic architecture which is influenced by popular big data processing Lambda architecture, based on distributed computing platform. The architecture using open-source platform Apache Flink for doing data processing. Flink is a popular platform for processing historical and stream data flows at once parallelly. Its stateful streaming can obtain more scalability and flexibility along with high throughput and low latency than the remaining stream processing programming models.

**Keywords** Big Data · Big Data Processing · Stream Computing · Apache Flink

## 1 Introduction

Over the past few decades, there has been an abnormal increase in the growth rate of data every year. The world is progressively becoming digitalized in various fields and sectors. Industries are actively seeking valuable insights from data to enhance their assets and

✉ P. Venkata Krishna
parimalavk@gmail.com

B. Gnana Deepthi
gdeepthi.bitra@gmail.com

K. Sandhya Rani
sandhyaranikasireddy@yahoo.co.in

V. Saritha
vsaritha@spmvv.ac.in

[1] Department of Computer Science and Engineering, Sri Padmavati Mahila Visvavidyalayam, Tirupati, India

tackle challenges in a competitive landscape. Many business sectors are adopting innovative technologies such as Cloud Computing, Artificial Intelligence (AI), Big Data Analytics, Internet of Things (IoT), and Machine Learning [1]. The primary objective of these advancements is to develop recommendation systems, personal assistant programs, self-driving vehicles, automation, and fraud detection systems, among others. However, managing the rapidly expanding and diverse data has become an intricate and demanding task for data administrators [2].

For performing complex data computations, centralized computing systems have been employed. In such scenarios, a central server system oversees the hardware and software components of multiple client systems and conducts computations. This approach of parallel processing proves cost-effective as the infrastructure is equipped with high-end hardware capable of handling substantial data volumes. To overcome challenges related to parallel processing, distributed and cloud computing technologies have been introduced [3]. These technologies facilitate dynamic online resource allocation and sharing, thereby enhancing system performance, scalability, reliability, and cost-effectiveness.

In the present era, real-world business applications predominantly opt for distributed systems with cloud infrastructure [4]. Companies are leveraging both historical and real-time data to extract valuable insights from the vast data influx, enabling the creation of modern applications for business expansion. The analysis of such data significantly contributes to organizational profits. Real-time data holds particular importance for application analysis. Systems must effectively respond to user queries and provide efficient customer services by promptly adapting to real-time circumstances. Online users represent a crucial asset for digital applications and organizations. Continuously evaluating and analyzing data falls within the realm of stream processing. This approach handles "real-time" data and continuously addresses queries to provide timely customer services [5].

A data stream [6] is a unique form of dynamic real-time data characterized by infinite unbounded variations. These data streams are derived from diverse sources and stored as compact data records in extensive clusters simultaneously. The primary processing methods for data streams are batch processing and micro-batch processing, which are based on time intervals. Predicting outcomes from such data streams for real-world decision support presents a considerable challenge. The expanding data landscape introduces new obstacles to extracting meaningful insights. Major cloud vendors offer virtual stream pipelines to process a range of data streams [7] (Figs. 1, 2, 3, 4, 5 and 6).

The realm of real-time big data analytics employs a variety of technological tools and architectures. Ongoing research is dedicated to analyzing real-time streaming data. These

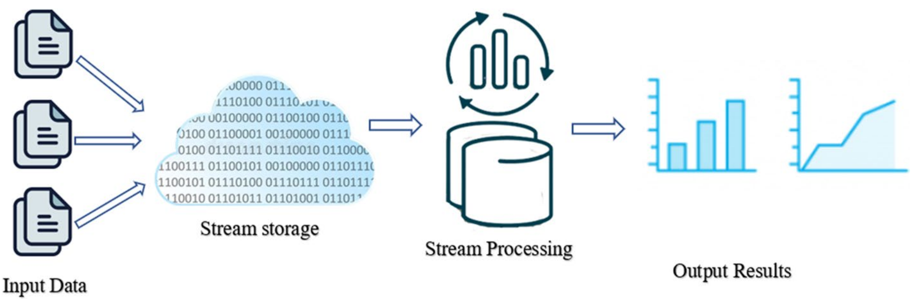


**Fig. 1** Data stream processing

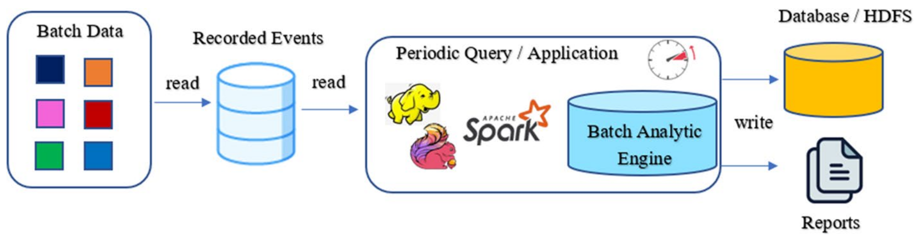


**Fig. 2** Batch processing

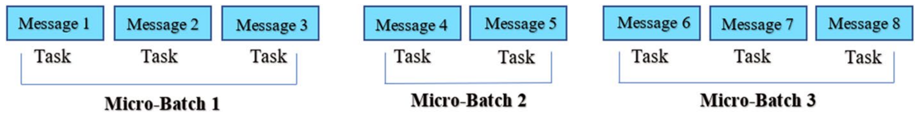


**Fig. 3** Micro-Batch processing

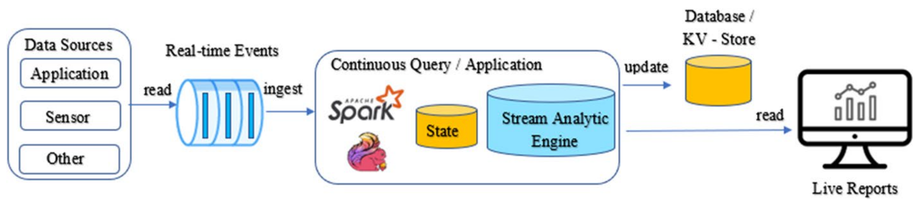


**Fig. 4** Stream processing

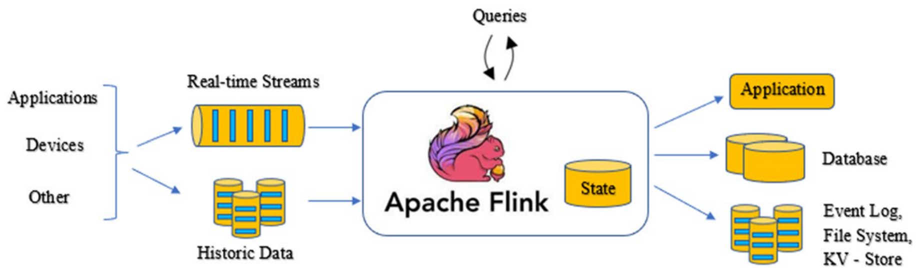


**Fig. 5** Apache flink platform architecture

studies indicate a wide array of tools being used for data processing, with no single tool serving as a definitive solution. Previous research relied on combinations of diverse tools to process data in both batch and real-time streams. However, integrating multiple tools introduces complexity, particularly for handling intricate datasets. Developing highly scalable and fault-tolerant systems is proving increasingly challenging.

To address these concerns, the focal contributions revolve around proposing a comprehensive architecture known as FRTSPS (Flexible Real-Time Traffic Stream Processing System) for analyzing real-time simulations of traffic data presented as extensive data streams. The term "data stream" here refers to the records captured by traffic cameras. FRTSPS

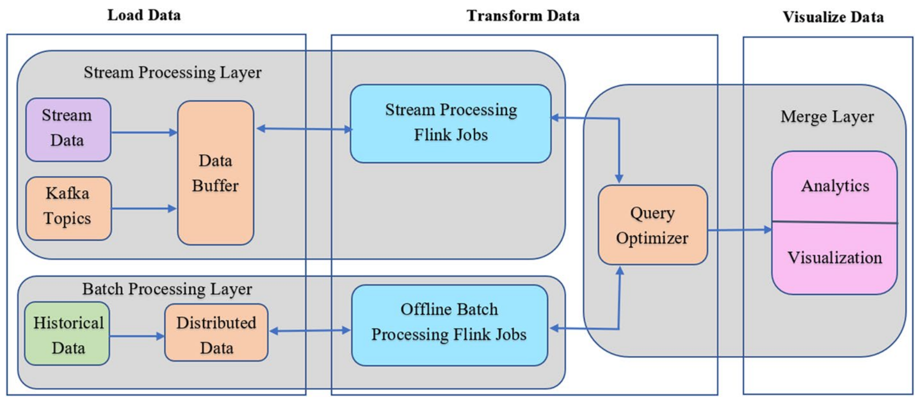


**Fig. 6** Working phases of proposed architecture FRTSPS

adopts a distributed architecture rooted in a popular data stream processing framework, catering to both real-time and batch-oriented processing needs. This architecture features distributed storage for bounded and unbounded data, data processing units, and a query optimization process to analyze vehicle counts recorded at specific time stamps.

The rest of this paper is structured into specific sections as outlined below. Section 2 furnishes background information pertinent to the research domain. Section 3 highlights previous work related to the subject. In Section 4, the architecture FRTSPS is introduced, detailing its design and system implementation. Notably, a segment of the model handles batch processing for historical data, while the remainder manages stream processing for real-time data streams. Section 5 elaborates on the key evaluation metrics employed and showcases the outcomes achieved through the utilization of FRTSPS. Finally, Section 6 encapsulates the conclusions drawn from this research endeavor.

## 2 Back ground

Data has become an integral part of our lives, with real-time applications and various domains generating substantial volumes of data. For instance, data originates from sources such as IoT devices [8], user interactions on the web, financial transactions, and location tracking systems, among others. A significant proportion of this data is characterized by high speed, heterogeneity, and streaming nature. It often lacks uniform organization and is in high demand, leading to complex processing challenges associated with these extensive datasets.

The term "big data" refers to an immense influx of data (volume) that grows rapidly (velocity) from diverse digital sources (variety), utilized for knowledge extraction and decision-making purposes [3]. Gartner [9] categorizes big data based on its initial three fundamental attributes, to which additional characteristics have been appended over time. These extensions encompass Volume (data of substantial size), Velocity (data with swift speed), Variety (data of assorted types), Variability (data with varying meanings and formats), Veracity (data with reliability and accuracy considerations), Value (data with significance), Visualization (data presented in diverse ways), Validity (data with defined consistency), Vulnerability (data prone to significant breaches), and Volatility (data with extended availability) [10, 11]. These attributes encapsulate the challenges posed by big data, complicating the three major phases of data storage, processing, and organization. Various tools and technologies have been devised to address these challenges.

In recent years, the analysis of big data [11] has emerged as a powerful approach aimed at devising architectures and technological models capable of effectively managing large datasets and supporting efficient decision-making systems across diverse real-time domains. These domains include engineering, science, healthcare, social media, anomaly detection, business intelligence, finance, marketing, recommendation engines, security, and fraud detection. Big data surpasses the scope of traditional database management systems. Similar to conventional mining algorithms, big data not only processes locally stored data but can also manage data spread across remote locations such as cloud servers [8], all while accommodating diverse data types. The organization and analysis of such extensive data are typically carried out in a distributed or streamed manner. Programming models like message passing interfaces are inadequate for handling big data. As a result, parallel processing offers an improved approach to enhancing processing performance.

Advancements in technology, along with supportive architectures and cloud computing platforms, have been developed to address the challenges of big data [7]. Novel programming models like MapReduce [10] have been introduced to handle large data volumes on commodity clusters at a substantial scale. Prominent big data frameworks such as Hadoop [11], Pig [13], Hive [2], and Mahout [12] are based on the MapReduce model. The appeal of MapReduce lies in its simplicity, parallel processing capabilities, cost-effective distributed nature, and robust fault tolerance, contributing to its swift adoption.

The distributed storage and processing inherent to the MapReduce model confer advantages over traditional data processing techniques. Yet, deploying data into computer hardware becomes increasingly complex when faced with extensive data inflow. However, the use of big data frameworks based on the MapReduce paradigm has encountered challenges related to performance and reliability due to the inability to reuse data effectively [12]. Job execution within the MapReduce framework involves data reading, writing, and MapReduce job execution, which can be slow due to latency-associated disk input–output operations [13]. This underscores the need for novel tools and techniques to tackle these issues and automate the storage and processing of massive datasets within supporting models.

### 2.1 Batch processing vs Micro-Batch processing vs stream processing

Big data processing involves three primary paradigms: Batch, Micro-batch, and Stream processing. In Batch processing, data is organized into finite bounded jobs. Large data volumes are collected as a batch within specific time intervals and subsequently undergo processing using a data analytical engine. The processing duration for this method tends to be longer, and jobs are executed according to predefined schedules. This paradigm is typically chosen when immediate processing results are not necessary [12].

An apt example of batch processing is observed in mainframe computers. Due to their involvement with substantial volumes of data, these systems encounter time-consuming data access and integration procedures that render continuous streaming unfeasible. Batch processing grapples with challenges such as fault tolerance, extended processing times, and cost-effectiveness [13].

The micro-batch processing approach serves as a midway point between batch and stream processing. In this method, data is organized into smaller groups known as micro-batches, with predetermined time intervals. Unlike batch processing, which accumulates all historical data into a single large batch over specific time increments, micro-batch processing involves smaller data groups being loaded and processed more frequently, adhering to pre-established thresholds [14]. This technique proves particularly valuable for scenarios requiring real-time data processing, such as online dashboards, server logs, and intrusion detection, as it strikes a balance between processing speed and data volume.

Stream processing involves executing jobs immediately upon the arrival of data within an infinite timeframe. This paradigm lacks a definitive endpoint for job execution. Data is continuously fed into the system as an uninterrupted flow, its size remaining uncertain. This approach entails real-time data analysis, enabling rapid insights. However, maintaining stream processing applications proves to be a complex task due to their continuous operation as long as data streams persist [15]. Stream processing is ideally suited for applications necessitating instant outcomes, offering shorter job execution times in comparison to batch processing.

Key attributes of stream processing encompass assured job execution, robust fault tolerance, periodic checkpointing to capture stream state, state information management, heightened throughput, minimal latency, and remarkable scalability. Noteworthy challenges in stream processing pertain to storage and processing considerations.

### 2.2 Comparison of big data processing models

This segment aims to provide a comprehensive comparison among the primary types of big data processing models: batch, micro-batch, and stream processing. By evaluating various parameters relevant to the organization and analysis of significant data across diverse platforms, we can assess performance metrics and challenging attributes for each of these programming models. As depicted below, Table 1 offers an overview of the comparisons.

In batch processing, data is stored for several hours or even days before undergoing analysis. This programming paradigm relies on the MapReduce model and is particularly suited for handling extensive datasets. Consequently, the processing performance tends to be slower compared to other models [16]. Apache Hadoop stands out as the leading framework for executing batch processing tasks [1].

Turning to micro-batch processing, this method involves the analysis of smaller-sized batches of nearly real-time data, typically arriving within minutes or seconds of specific time intervals. This approach demonstrates swifter performance than batch processing yet slightly lags behind stream processing. Notably, it accommodates a wider range of programming languages. Apache Spark [17] streaming emerges as a prominent platform for micro-batch processing, attributed to its in-memory caching capabilities. Data is directly transmitted to the streaming analytical engine without prior storage, making it an online dynamic process that boasts advantages over alternative processing models.

Apache Flink [16] has gained traction as a favored platform for stream processing, offering integration with various platforms and frameworks. Starting as an incubating project under the Apache Software Foundation in April 2014, Apache Flink has garnered widespread adoption by contributors and business entities across the globe. It adeptly processes bounded and unbounded data streams in a distributed, scalable, and fault-tolerant manner. Furthermore, Flink [18] can be characterized as a platform rather than a mere framework due to its robust integration capabilities with other frameworks. Operating as a distributed stream processor, Flink provides diverse APIs at varying levels of abstraction to facilitate stateful stream processing, accompanied by specialized libraries catering to different application scenarios. The Dataset abstraction API serves batch processing purposes, while the

**Table 1** The comparison of the big data processing models [1, 3, 4, 12, 14, 16]

| Parameters | Batch Processing | Micro-Batch Processing | Stream Processing |
|---|---|---|---|
| Data Storage | Data within an hour or days (Historical data) | Data arrived at recent real-time | Recently arrived data at sub-second timeframes |
| Job Implementation | Finite job (batch) | Small group of batches | Data streams |
| Frameworks supported | Apache Hadoop, Apache Spark, GraphX, Alteryx | Apache Spark Streaming, Vertica, Fluentd, Logstash | Apache's Spark, Storm, Flink, S4, sFlume, Confluent's KSQL, Samza |
| API Languages | Java | Java, Scala, R, Python | Java, Scala, Python |
| Stateful operations | No | No | Yes |
| Batch / Stream primitive | MapReduce | Dstream | DataStream |
| In-memory cache | No | Yes | Yes |
| Optimization | Manual optimization | Manual optimization | Having independent query optimizer |
| Time delay | Huge data processing at once causes delay | Relatively faster because of in-memory caching | Very Faster about in milliseconds |
| Processing speed | Slower | Faster than batch | Faster than micro-batch |
| Throughput | Low | High | High |
| Latency | High | Low | Low |
| Fault tolerance | Partially fault tolerant | Not fully fault tolerant because of in-memory caching and Resilient Distributed Datasets | Distributed snapshots for each job execution ensures Fully fault tolerance |
| Security | Kerberos authentication is somewhat difficult | Allows Kerberos authentication | Allows Kerberos authentication |
| Scalability | High | Very high | Very high |
| Performance | Slower | Not as efficient | Efficient |
| Cost | Not Cost-effective since done on commodity hardware | Cost-effective since requires high RAM | Cost-effective since requires high RAM |
| Real-time analysis | Fails in real-time analysis | Supports real-time analysis | Supports real-time analysis |

DataStream [3] abstraction API is employed for stream processing. Programming tools supported for data processing in Flink encompass Java, Scala, Python, and R [4].

Flink [19] is an Open-source true stream processing tool majorly can process both batch and stream data. Query processing is two to three times faster than the other stream processing frameworks because of its query optimizing engine and can provide high throughput and low latency. In-memory storage is automatic and scalable. The processing operations map, filter and reduce are the continuous long running operations in the framework.

## 3 Related work

In robotic applications, rapid human face recognition is vital for improved human–machine interaction. Cloud technology integration mitigates speed limitations by dynamically providing memory and processing power. However, public cloud usage raises concerns about image database security. To address this, we explore secure face recognition options where recognition occurs on encrypted data. We evaluate eight encryption algorithms for security and recognition accuracy. Face recognition tests in robot and cloud environments measure average recognition time using Principal Component Analysis (PCA) on encrypted data. This research aims to enhance secure face recognition's feasibility and efficiency in practical applications [33].

This paper outlines our current endeavors in creating the E-Recall system, an innovative platform designed for cloud-based mobile rich media data management. The primary objective is to establish an intelligent and all-encompassing infrastructure that addresses (1) Enabling efficient processing of media data at scale. (2) Facilitating flexible sharing and publication of media content. (3) Supporting personalized integration of media content within a mobile environment. The E-Recall system represents a pioneering approach to addressing these multifaceted challenges in media management and mobility [34].

Van Dongen et al. [17] discussed about the scalability of stream processing jobs in the view of performance and efficiency. The frameworks denoted are: Flink, Kafka Streams, Spark Streaming, and Structured Streaming. Horizontal and vertical scaling issues are evaluated. Scaling considering no of factors like framework design, resource allocation, pipeline design and data characteristic. Previous works focus on evaluating latency, throughput, fault tolerance and scalability factors individually through their frameworks. Here a proposing model. OSPBench suite is used to evaluate additional efficiency of the same factors with in a single suite. Data streams considered are taken from Netherlands national traffic data. OSPBench suit design for finding efficiency of workload factors and used elasticity to improve the scalability of stream processing. For large clusters, there should be researches required to increase the performance when large pipelines considered.

Iwendi et al. [20] implemented an architecture to process and analyze heterogeneous data streams from various digital IOT devices. This author considered datasets of Newsgroups for text analysis. Proposed architecture mainly focused on result accuracy and execution time. The partition based and mining-based algorithms are used for text analysis. Louvain method is used for detecting and extracting separate words from the input data. Input streaming data processed with map reducing and accurate results are obtained by extracting the knowledge from the data. Text analysis of volumes of input data resulting efficient outputs if the input is increased. Streaming data complex structure reduced to simple structure.

The applications of big data in healthcare sectors provides quality of services with cost effective advancements. Big data stream computing is a big asset for this domain for doing effective analyzation of rapid growing health data. Ta et al. [21] introduced a novel architecture for the prediction and analyzation of health records by using various big data analytical tools. The architecture is generic combination of various big data tools like Apache Kafka, Storm and NoSQL Cassandra.

Data streams collected from various medical sources, stored and processed in a distributed architecture. Proposed architecture can do batch and stream computing of health care records. Through the stream computing, applications get efficient storage and query optimization. Additional benefits with the analysis of streams are fraud detection, drug discovery, medical diagnosis and support systems. Due to the inconsistency of medical data, the analysis is a difficult job for the processing of real-time data streams. Classical data mining tools give transparent results for such case but if additionally, machine learning applications reduce the complexity of the distributed processing of data streams.

For the real-time heterogeneous data, batch processing is not adequate whereas the data is time-variant and require efficient prediction models. Akanbi et al. [22] explores a distributed middleware framework for stream processing of heterogeneous real-time data using various big data technologies. Apache Kafka having greater potential for processing datasets from local machines and sensor devices. Effective Drought Index (EDI) model predicts data streams in two steps. Running the predictive model and querying infinite streams are the two processing steps involved in this framework.

Dynamic analysis and real-time processing of moving objects data sets is a challenging job for the development of algorithms. Spatial concentrations are used for detecting the traffic jams or crowd population on the busy roads or junctions. For these kinds of issues most of the applications using batch processing scenarios where traffic positions of moving objects considered in timely manner and batch processed. Real-time streaming data applications demands stream processing for faster results. Roriz Junior et al. [23] proposed a parallel algorithm, DG2CEP which uses DBSCAN algorithm to process streaming data with low latency. Streaming data complex structure reduced to simple structure. Rush hours of traffic dynamically changes and the system need to accommodate the input streams and process dynamically.

Health care Sector having vast advancements regards to big data scenarios. The analysis and prediction of health reports, helps the medical practitioners for giving efficient services to the patients. Actually, the health reports are collected from various data sources. Vanathi et al. [24] explores the applications of big data stream processing in health care sector. The proposed architecture in this paper is a combination of various big data tools Apache Storm, Apache Kafka and NoSQL Cassandra. The architecture can process in both batch and stream processing manner and is scalable.

Disaster management is the one of the real-time applications where big data and Cloud computing emerges for analysis useful in emergency situations. Puthal et al. [25] proposed a framework for predicting real-time emergency decision support services and alert generation. For such kind of applications user's sensitive data must be protected. This paper majorly discussing about the security aspects of the data streams. Real-time processing of sensitive data is a challenging job. Apache Storm open-source tool is used for implementing alert systems and predicting the disaster areas.

Internet of things having emerging applications in the big data domain. The data transmission from digital devices is dynamic and heterogeneous in nature. Corral-Plaza et al. [26] proposing an architecture that preprocess the data streams by homogenizing them for better query optimization. The architecture is a generic one and is used for various sources of IOT devices. Here author considered the real-time data stream like water supply management data that is processed, transformed and further analyzed by the data processing tools Kafka, Apache Avro and Esper.

Stream processing frameworks solving complex data organizing jobs in the field of parallel processing. Scalability, Reliability and Fault tolerance are the major advances of such kind of processing scenario. Van Dongen et al. [27] worked on the comparison of the big data stream processing tools Spark, Flink and Kafka. The major aspects considered here are the storage failures and job failures. The job scheduling approach between the tools given efficient results whereas Storm framework having better results.

## 4 Flink based real-time stream processing system (frtsps)

Existing big data tools rely on batch processing techniques with high latency, which proves inadequate and time-consuming for processing critical applications and real-time streaming data [22]. Batch processing methods handle bounded data periodically, yielding results in a time-ordered fashion. Conversely, stream processing techniques cater to both bounded and unbounded data, providing real-time results in a matter of seconds, contingent on stream velocity. However, the accuracy of stream processing outcomes may fall short compared to batch processing due to the larger data volume processed.

To overcome these challenges, the lambda architecture [28] emerges as a solution by integrating batch and stream processing techniques to deliver enhanced and precise results. Our proposed architecture, FRTSPS, is built upon the lambda architecture [29], featuring three layers: the batch processing layer, the stream processing layer, and the service layer. The stream processing layer adeptly manages incoming real-time streaming data, while the batch layer captures substantial volumes of bounded data at specific time intervals. This data is stored in a distributed manner and analyzed using the map-reduce technique. The service layer acts as an intermediary connecting the batch and stream layers, facilitating the visualization of analytical results as output. In essence, the lambda architecture synergizes both batch and stream processing layers to effectively analyze historical data and real-time data streams [30].

Within FRTSPS, the focus is on the real-time traffic dataset, which is organized into distinct categories based on location and time windows. This dataset encompasses various attributes, including vehicle ID, vehicle speed, position, and traffic light data. The primary objective involves simulating the dataset to ascertain the volume of vehicles traversing specific locations during designated time intervals.

As an example, we present a straightforward algorithm that outlines the control logic employed within the system implementation.

**Algorithm to simulate Real-time Traffic Data in FRTSPS**

```
def Vehicle Model
1. # Load Data
2.     # The producer service
3.         build a pipeline to the records Vehicle Plate No, Timestamp, Vehicle type.
4.         write the data to Kafka topics to randomly generate vehicle events.
5.         handle the topic messages at local cluster with topic name "vehicle-type."
6.         partition the data based on the vehicle type
7. # Transform Data
8.     # The Consumer Service
9.         map the results from producer by serialization.
10.        assign keyed data stream to windows operator.
11.        aggregate the sum of vehicles crossed at the specific time frame.
12.        deserialize the Vehicle model.
13. # Visualize Data
14         eliminate conflicts and duplicates
15         analyze the data
16         return vehicle count crossed at the certain time stamp.
17         compute throughput and latency in terms of vehicle count
18         count number of Vehicle events within a multi-broker cluster
19         Estimate System runtime ratio
End
```

### 4.1 Load data

During this phase, the data transmissions are bifurcated into two distinct pipelines to manage both streaming and batch data. The Batch processing layer, operating within the first pipeline, handles incoming data at specific time intervals. This data is subsequently stored in a distributed database engine for further processing. Should new data arrive within the same time window, it's appended to the existing master data in the distributed storage system. On the other hand, the stream processing layer captures real-time data, with varying data stream speeds. To manage the fluctuating load, Apache Kafka [21] topics are leveraged.

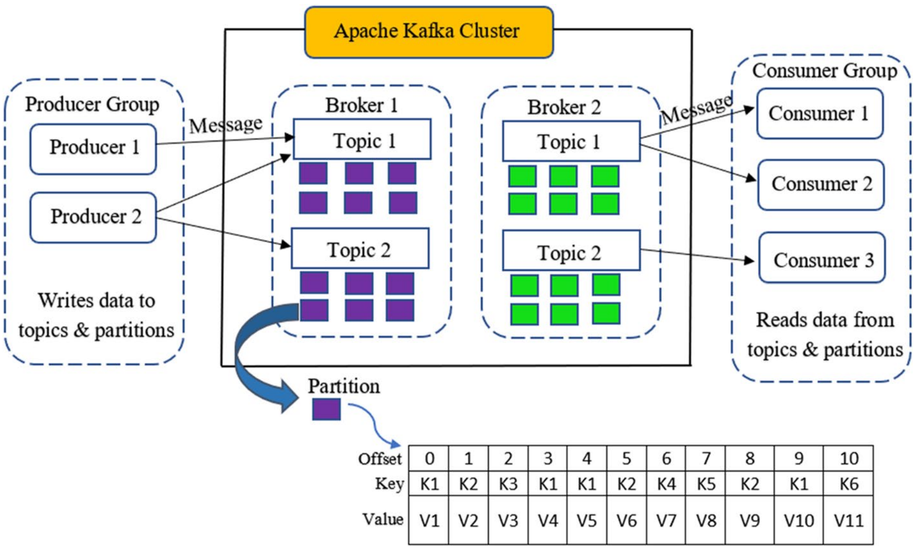
Kafka [22] functions as a data buffer, effectively moderating the flow of streaming data at the analysis level. Input data originating from digital sources is managed through two topics within the framework: producers and consumers. Producers extract data from pipelines and transform it for analysis by consumers. This mechanism acts as a channel between the traffic system and the sensors. Streaming data is integrated into the system in message format, denoted as a topic. Each message contains two key parameters: producers and consumers, each specialized in handling distinct data types. Incoming data is managed by producers and analyzed by consumers. Topics serve as the mechanism for organizing data, where producers send data to topics, and consumers retrieve data as messages. The number of topics depends on data size, including attributes like vehicle speed, position, and traffic lights. Both structured and unstructured data can be accommodated [24].

A Kafka cluster [26] oversees each topic with a unique identifier. Producers write data to topics, and consumers send messages to specific topics for data retrieval. Each topic can be subscribed to by zero or multiple consumers. Broker nodes within the cluster handle partitioning and replication of topics to achieve high throughput. Every message is assigned a unique offset value to facilitate easy tracking. Topics can be created manually or automatically. The process of organizing topics involves four sequential steps: listing topics, describing topics, creating topics, and deleting topics. The cluster layout is visually represented in the Fig. 7.

## 4.2 Transform data

This phase is designated for data transformation, achieved through specific operations. Within the batch processing layer, data is subjected to the Extract, Transform, and Load (ETL) process [22]. The outcome of this process constitutes batch views that are subsequently indexed. These batch views can be scaled for further indexing when new ones become available in the pipeline. Due to the sheer volume of data, this process is relatively slow. However, the stream layer compensates for this time delay by processing real-time data streams. This layer is responsible for generating real-time views through the processing of recently arrived data that hasn't undergone transformation in the batch layer.

In the transform layer, a Kafka Consumer is utilized within the Apache Flink [31] job. Its role is to provide statistical results pertaining to specific vehicle types crossing predefined speed thresholds within certain time intervals. Kafka [11] data nodes are employed for storing stream data, while PostgreSQL tables accommodate bounded data. The Consumer collects messages and interprets them as data streams. Flink [4], utilizing a master–slave processing model, initializes the job manager for task scheduling on worker nodes, generating individual jobs for each data stream event. Task managers oversee these events, issuing heartbeat signals to nodes throughout processing. Stream abstraction is



| Offset | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| Key | K1 | K2 | K3 | K1 | K1 | K2 | K4 | K5 | K2 | K1 | K6 |
| Value | V1 | V2 | V3 | V4 | V5 | V6 | V7 | V8 | V9 | V10 | V11 |

**Fig. 7** Apache kafka cluster with producer, consumer groups and partition layout

executed using DataStream objects [10], which receive data from the cluster and parallelly perform map, filter, and reduce operations on the data for the respective job events.

Data retrieval from the PostgreSQL database is facilitated by the JDBC connector, forwarding it to the phase's query optimizer. Here, batch and stream views are merged. To ensure fault tolerance, snapshot checkpoints are implemented in the event of node, hardware, software, or network failures. In such cases, DataStreams [3, 5] are automatically restarted, and checkpoint log values are reset.

The Query optimizer phase involves parallel data processing in both stream and batch formats. Notably, both layers contain data in the same format. Should any discrepancies arise in batch or stream views, the query optimizer reevaluates the views as new data is appended to existing records. Data cleaning ensues, addressing error values to minimize inaccuracies. The transform phase consistently queries the latest available data. Windowing and SQL transformations [32] are also employed to eliminate duplicates. Streaming windows partition infinite data streams into finite-sized buckets for processing, guided by certain parameters. Following these operations, the transformed data materializes as a SQL-ready timestamp.

### 4.3 Visualize data

The final phase within FRTSPS is dedicated to visualization, with its primary objective being to present the analyzed data to end-users. This phase recognizes that the batch layer is designed for historical data, while the stream layer caters to real-time streaming data. Consequently, both the batch and stream layers simultaneously transmit data to this visualization layer in the form of metrics for analysis.

Data is written to the merge layer, facilitating the merging of streaming and batch data concurrently. Subsequently, the results from both layers are combined while eliminating any conflicts or duplicates. The processed data is then made readily available within the database for analysis. To achieve this, we employ the Cassandra sink, which outputs processed streams to the same Cassandra table. The outcomes generated in the visualization phase manifest as analytical dashboards for users to interpret and derive insights from.

## 5 Results & discussion

The underlying goal behind introducing the generic architecture FRTSPS in this paper is to seamlessly integrate both batch and stream processing programming models. We delve into the fundamental concepts of big data processing, which play a crucial role in efficiently analyzing extensive datasets in real-time. The study encompasses various data types and sizes, which are relevant for stream analytics. Through a comprehensive evaluation, diverse parameters are scrutinized to gauge the system's efficiency and effectiveness. These parameters offer insights into the system's performance within designated time frames and its responsiveness to generic real-time queries. FRTSPS lays out a flow-based, generic analytical model, serving as the foundation for data storage and management.

In Section 4, we present the system model and elaborate on the flow-based analysis of stream processing across multiple layers. The major phases within the system encompass data loading, data processing, and the merging of in-flow stream analysis. The data cluster exhibits scalability to accommodate varying data bandwidths while effectively managing faults that may arise in individual machines. In-memory capabilities aid in processing large

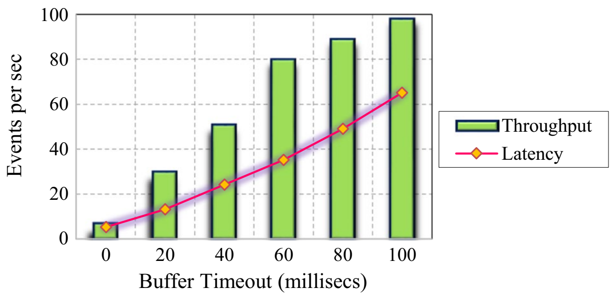


**Fig. 8** Variations in throughput and latency as the buffer timeout increases

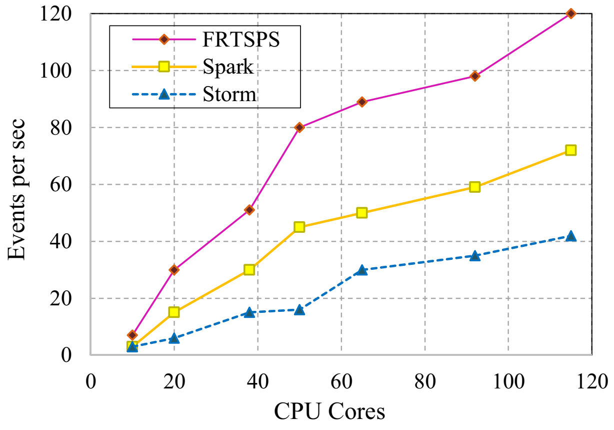


**Fig. 9** Comparison of throughput for frameworks Spark, Storm and FRTSPS

volumes of data streams effortlessly, capable of handling diverse data simultaneously. During the data analysis phase, the query optimizer is implemented through reducer and evaluator modules. These modules employ aggregation functions to count vehicles at specific time stamps.

The map process is integrated into the producer phase, aligning with the real-time data flow. Through an evaluation, we measure performance metrics such as throughput and latency for our FRTSPS architecture. Total throughput and latency for each job are assessed concerning buffer timeout and the number of events processed per second. The execution of jobs involves 20 task managers. All metrics are recorded, leading to the calculation of average throughput and latency for our data.

Figure 8 illustrates the comparison between average throughput and latency for a specific number of events. As we extend the batch interval by a few seconds, throughput demonstrates an increase. Correspondingly, inherent latency becomes apparent. This observation implies that event-driven processing within the clusters necessitates limited buffers. Notably, a sustainable throughput of 9.82 MBS/s/event is achieved. The visualization of performance metrics further confirms that FRTSPS is capable of delivering high throughput and minimized latency.

Figure 9 presents a comparative analysis of FRTSPS's throughput results in relation to other frameworks. The data points, depicted as dot markers, showcase throughput as the chosen performance metric, specifically considering vehicle events within a multi-broker cluster. The evaluation encompasses the number of events per cycle and the count of

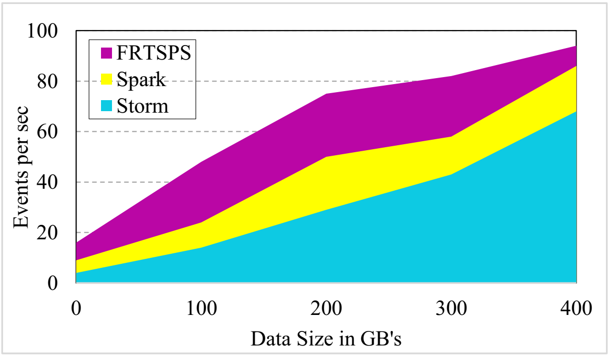


**Fig. 10** Processed events per second

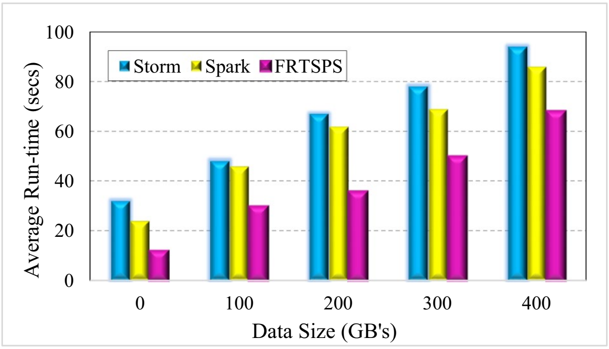


**Fig. 11** System Run-time ratio

CPU cores employed per job execution. Notably, it is evident that FRTSPS yields higher throughput outcomes as both the number of events and CPU cores increase.

In Figs. 10 and 11, we delve into a comparison involving processed events per second and the system run-time ratio for FRTSPS, particularly juxtaposed with platforms Spark and Storm. Within this context, Apache Flink emerges with notably higher throughput in comparison to the other platforms. Notably, its behavior follows a linear trajectory and maintains a high saturated ratio, underlining its robust performance characteristics.

## 6 Conclusion

In this work, FRTSPS proves to be robust, reliable, scalable, and elastic for processing data streams. It comprises three fundamental phases: pre-processing, transformation, and analysis, and demonstrates the capability to handle substantial real-time streams. Moreover, it seamlessly integrates both batch and stream processing for complex datasets. Notably, the query optimizer exhibits robustness in addressing intricate queries as each topic is designed to manage distinct types of data.

However, it’s essential to acknowledge certain limitations, primarily related to the processing of extensive real-world data streams. The evaluation of multi-level queries is facilitated through the utilization of simple Python functions.

Real-world applications necessitate the analysis of heterogeneous data streams rapidly generated from various digital sources. Conventional data processing tools and applications fall short in effectively addressing such challenges. Consequently, there arises a demand for the development of intricate data structures to represent processing and networking architectures. To complement this, further research is essential for refining existing big data tools and implementing more advanced analytical models, such as machine learning architectures.

**Funding** The authors have not disclosed any funding.

**Data availability** The datasets generated during and/or analysed during the current study are available from the corresponding author on reasonable request.

## Declarations

**Conflicts of interests** The authors declares that there is no conflict of interest for this paper.

## References

1. Kiran M, Murphy P, Monga I, Dugan J, Baveja SS (2015) Lambda architecture for cost-effective batch and speed big data processing. 2015 IEEE International Conference on Big Data (Big Data), Santa Clara, CA, USA, pp. 2785–2792. https://doi.org/10.1109/BigData.2015.7364082
2. Isah H, Abughofa T, Mahfuz S, Ajerla D, Zulkernine F, Khan S (2019) A survey of distributed data stream processing frameworks. IEEE Access 7:154300–154316
3. Tantalaki N, Souravlas S, Roumeliotis M (2020) A review on big data real-time stream processing and its scheduling techniques. Int J Parallel Emergent Distrib Syst 35(5):571–601
4. Lopez MA, Lobato AG, Duarte OC (2016) A performance comparison of open-source stream processing platforms. 2016 IEEE Global Communications Conference (GLOBECOM), Washington, DC, USA, pp. 1–6. https://doi.org/10.1109/GLOCOM.2016.7841533
5. Rabl T, Traub J, Katsifodimos A, Markl V (2016) Apache Flink in current research. It-Inform Technol 58(4):157–165
6. Feng L (2020) A real-time computer network trend analysis algorithm based on dynamic data stream in the context of big data. 2020 International conference on intelligent transportation, big data & smart city (ICITBS), Vientiane, Laos, pp. 473–476. https://doi.org/10.1109/ICITBS49701.2020.00102
7. Carbone P, Fragkoulis M, Kalavri V, Katsifodimos A (2020) Beyond analytics: The evolution of stream processing systems. In Proceedings of the 2020 ACM SIGMOD international conference on management of data (SIGMOD ’20). Association for computing machinery, New York, USA, 2651–2658. https://doi.org/10.1145/3318464.3383131
8. Marques, Nuno C, Bruno Silva, Hugo Santos (2016) An interactive interface for multi-dimensional data stream analysis. 2016 20th International Conference Information Visualisation (IV), Lisbon, Portugal, pp. 223–229. https://doi.org/10.1109/IV.2016.72
9. De Mauro A, Greco M, Grimaldi M (2016) A formal definition of Big Data based on its essential features. Libr Rev 65(3):122–135
10. Carbone P, Katsifodimos A, Ewen S, Markl V, Haridi S, Tzoumas K (2015) Apache flink: Stream and batch processing in a single engine. The Bulletin of the Technical Committee on Data Engineering, 38(4):28–38
11. Jiang W, Luo J (2022) Big data for traffic estimation and prediction: a survey of data and tools. Appl Syst Innov 5(1):23
12 Nazari E, Shahriari MH, Tabesh H (2019) BigData analysis in healthcare: apache hadoop, apache spark and apache flink. Front Health Inform 8(1):14

13. Naoual El aboudi and Benhlima L (2018) Big data management for healthcare systems: architecture, requirements, and implementation." Advances in Bioinformatics, 2018(4059018):10. https://doi.org/10.1155/2018/4059018
14. Venkataraman S, Panda A, Ousterhout K, Armbrust M, Ghodsi A, Franklin MJ, Recht B, Stoica I (2017) Drizzle: Fast and adaptable stream processing at scale. In Proceedings of the 26th Symposium on Operating Systems Principles, 374–389. https://doi.org/10.1145/3132747.3132750
15. Fragkoulis M, Carbone P, Kalavri V, Katsifodimos A (2020) A survey on the evolution of stream processing systems. arXiv preprint arXiv:2008.00842
16. Mahapatra T (2020) Composing high-level stream processing pipelines. J Big Data 7(1):1–28
17. Van Dongen G, Van Den Poel D (2021) Influencing factors in the scalability of distributed stream processing jobs. IEEE Access 9:109413–109431
18. Shahverdi E, Awad A, Sakr S (2019) Big stream processing systems: an experimental evaluation. In 2019 IEEE 35th International Conference on Data Engineering Workshops (ICDEW), 53–60
19. HoseinyFarahabady MR, Jannesari A, Taheri J, Bao W, Zomaya AY, Tari Z (2020) Q-flink: A qos-aware controller for apache flink. 2020 20th IEEE/ACM International Symposium on Cluster, Cloud and Internet Computing (CCGRID), Melbourne, VIC, Australia, pp. 629-638. https://doi.org/10.1109/CCGrid49817.2020.00-30
20. Iwendi C, Ponnan S, Munirathinam R, Srinivasan K, Chang C-Y (2019) An efficient and unique TF/IDF algorithmic model-based data analysis for handling applications with big data streaming. Electronics 8(11):1331
21. Ta, V-D, Liu C-M, Nkabinde GW (2016) Big data stream computing in healthcare real-time analytics. In 2016 IEEE international conference on cloud computing and big data analysis (ICCCBDA), pp. 37–42. IEEE
22. Akanbi A, Masinde M (2020) A distributed stream processing middleware framework for real-time analysis of heterogeneous data on big data platform: case of environmental monitoring. Sensors 20(11):3166
23. Roriz Junior M, Olivieri B, Endler M (2019) DG2CEP: a near real-time on-line algorithm for detecting spatial clusters large data streams through complex event processing. J Internet Serv Appl 10(1):1–28
24. Vanathi R, Khadir AS (2017) A robust architectural framework for big data stream computing in personal healthcare real time analytics. 2017 world congress on computing and communication technologies (WCCCT), Tiruchirappalli, India, pp. 97–104. https://doi.org/10.1109/WCCCT.2016.32
25. Puthal D, Nepal S, Ranjan R, Chen J (2016) A secure big data stream analytics framework for disaster management on the cloud. 2016 IEEE 18th International Conference on High Performance Computing and Communications; IEEE 14th International Conference on Smart City; IEEE 2nd International Conference on Data Science and Systems (HPCC/SmartCity/DSS), Sydney, NSW, Australia, pp. 1218–1225. https://doi.org/10.1109/HPCC-SmartCity-DSS.2016.0170
26. Corral-Plaza D, Medina-Bulo I, Ortiz G, Boubeta-Puig J (2020) A stream processing architecture for heterogeneous data sources in the Internet of Things. Comput Stand Inter 70:103426
27. van Dongen G, Van Den Poel D (2021) A performance analysis of fault recovery in stream processing frameworks. IEEE Access 9:93745–93763
28. Hasani Z, Kon-Popovska M, Velinov G (2014) Lambda architecture for real time big data analytic. ICT Innovations 133–143
29. Probst L, Rauschenbach F, Schuldt H, Seidenschwarz P, Rumo M (2018) Integrated real-time data stream analysis and sketch-based video retrieval in team sports. 2018 IEEE International Conference on Big Data (Big Data), Seattle, WA, USA, pp. 548-555. https://doi.org/10.1109/BigData.2018.8622592
30. Qadah E, Mock M, Alevizos E, Fuchs G (2018) Lambda architecture for batch and stream processing. In CEUR Workshop Proc 2083:109–116
31. Li Z, Yu J, Bian C, Pu Y, Wang Y, Zhang Y, Guo B (2020) Flink-er: an elastic resource-scheduling strategy for processing fluctuating mobile stream data on flink. Mobile Information Systems, 2020(5351824):17. https://doi.org/10.1155/2020/5351824
32 Van Dongen G, Van den Poel D (2020) Evaluation of stream processing frameworks. IEEE Trans Parallel Distrib Syst 31(8):1845–1858
33. Karri C (2021) Secure robot face recognition in cloud environments. Multimedia Tools Appl 80(12):18611–18626
34. Shen J, Yan S, & Hua XS (2010). The e-recall environment for cloud based mobile rich media data management. In Proceedings of the 2010 ACM multimedia workshop on Mobile cloud media computing. 31–34. https://doi.org/10.1145/1877953.1877963