

Distributed Data Streaming Processing

Maitham Al-rubaye

Supervisor: Atakan Aral

Computer Science Department

University of Vienna

2023

First and foremost, I would like to express my profound gratitude to my parents. Their unending support, patience, and love have been the foundation upon which my academic and personal growth has been built. They instilled in me the value of hard work and perseverance, and for this, I am eternally grateful.

To my brother and sister, thank you for your constant encouragement and for always being there for me. Your belief in my abilities, even in moments of doubt, has been invaluable throughout this journey. Your companionship and camaraderie have made the challenges faced along the way much easier to bear.

Lastly, I would like to extend my heartfelt appreciation to my advisor, Professor Atakan Aral. His guidance, expertise, and mentorship have been instrumental in shaping this research. His dedication to my academic progress and his unwavering belief in my potential have not only made this thesis possible but also enriched my journey in the field of Computer Science.

This accomplishment would not have been possible without the unwavering support and love from all of you. I am forever thankful for having you in my life.

Table of Contents:

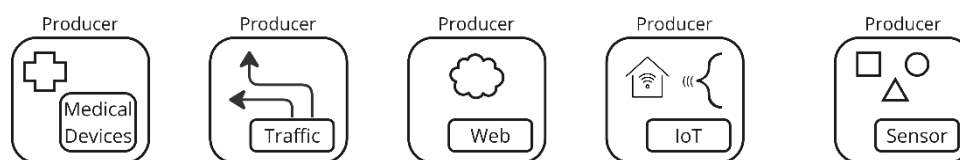
• Introduction	4
• Overview of Data Streaming	4
• Data computing concepts	5
• Distributed Data Streaming Processing	6
• Data Streaming Technologies	7
• Practical	9
- Objective, Data Generation, Infrastructure & Mechanism	9
- Flow, Experimentation, Observations, Results & Visualization	10
- paper Of the Results	11
- Concluding Thoughts	12
• GitHub links	12
• Resources	13

Introduction

The proliferation of real-time data has resulted in a significant increase in the demand for efficient data streaming processes. This study explores the concept of distributed data streaming, its importance, and the techniques used to implement it.

Overview of Data Streaming

Data streaming, as well, known as stream processing, is a set of methods used to process massive volumes of real-time or near-time data, quickly and continuously. Data streaming involves extracting, transforming, and loading (ETL) sizable sets of data in a continuous, and efficient manner. It is designed to handle high volumes of data coming in at high speed, such as real-time data from various sources including IoT, web, social media, sensors, traffic, medical devices, and many others.



To cut it closer, data streaming process comprises:

Continuous data generation: Data producers are various sources such as, IoT (internet of things), different purposes sensors, web and applications, healthcare, social media, and as elaborated before many others. These sources produce volumes of data in velocity, variety, and resulting in overcrowding bulk of data.

Real-time or near-real-time processing: Data streaming focuses on processing data once it generates, for the purpose of organizations to allow analyzation, and decision making to events in real-time or near-real-time. That is to be done even before storing the data.

Stateful and Stateless processing: Stateless processing treats each data point independently, whereas stateful processing maintains information about the history of previous data points to analyze and process the current data point.

Grouping: In stream processing, data is often grouped into time-based or count-based windows. This allows the system to process and analyze data within specific timeframes or data point ranges, making it easier to detect patterns, trends, and anomalies.

Scalability and fault-tolerance: Data streaming systems are designed to be scalable and fault-tolerant, ensuring that they can handle increasing amounts of data and recover from failures. This is typically achieved through distributed processing, replication, and checkpointing mechanisms.

Data storage and integration: Stream processing systems can store processed data in databases, or data warehouses for further analysis or reporting. Interoperability can also be achieved with other systems for data ingestion, processing, and storage.

Popular data streaming technologies include Apache Kafka, Apache Flink, Apache Storm, and Apache Samza. These technologies offer various features and capabilities for handling data streams, such as ingestion, processing, storage, and integration with other systems.

Data computing concepts

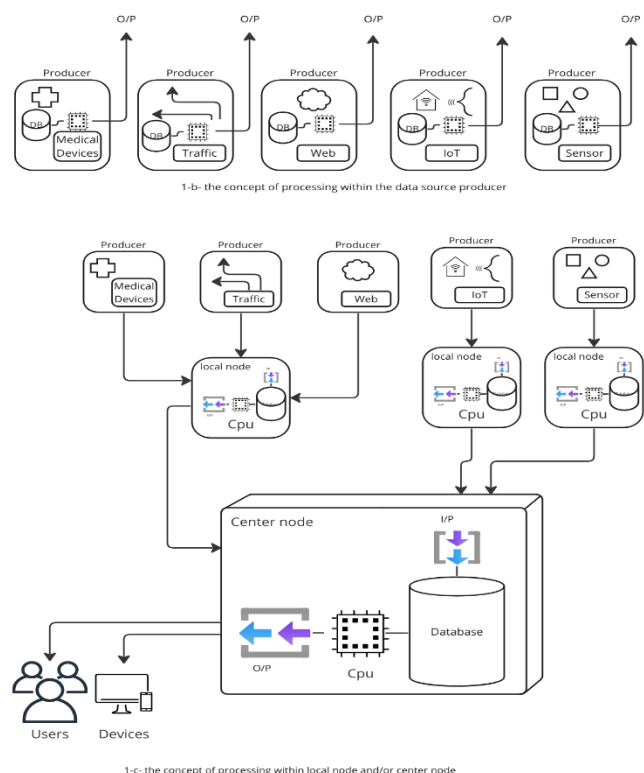
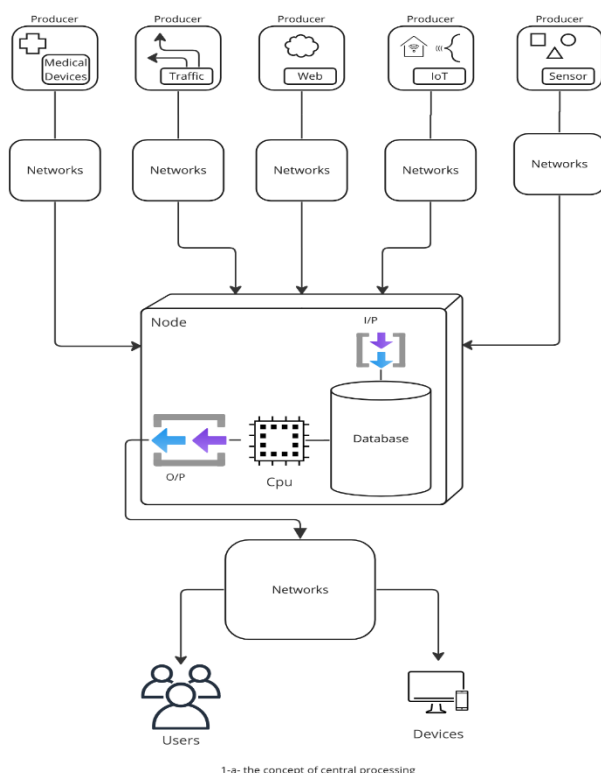
As processing data has become a key competence, several strategies have shown up, the most famous paradigms are cloud computing and edge computing. Moreover, there are several other paradigms, like Fog Computing, Grid Computing, Quantum Computing, Green Computing and High-Performance Computing. Each of these paradigms has its own set of uses and advantages, and the choice of which to use depends on the specific requirements of the task at hand.

The history of cloud computing started with the practice of time sharing in the 1950s. Time-sharing systems were developed to make better use of expensive mainframe resources by allowing multiple users to share these resources simultaneously. Then in the 1990s, telecommunications companies, who previously only used single dedicated point-to-point data exchanging, began using virtual private network services. Then, they were able to control overall network bandwidth more effectively. The term “cloud computing” appeared around 2006 when the cloud began using the term to describe the new paradigm in which the use of computer power, and accessing applications and files has started to be managed through the internet instead of using personal PCs.

Edge computing term is a more recent development, emerging around 2014, as a response to the limitations of cloud computing. As more devices became internet-connected and began generating vast blocks of data, it became impractical to send all that data to the cloud for processing due to latency and

bandwidth constraints. This concept involves processing data closer to where it is generated to reduce latency, save bandwidth, and for more privacy and security concerns.

It's important to note that while these approaches can be used separately, they can also be used together. For instance, an IoT system might use edge computing to process data on the device and then send the results to the cloud for further processing or storage. The drawings below illustrate different scenarios of the processing.



Distributed data streaming processing

Distributed data processing emerged as a reaction to the limitations of centralized computing, where a single mainframe, server or node was responsible for processing all data. As data volumes grew and computing technology advanced, it became more efficient and necessary to perform the processing tasks across multiple machines. That all leads to discuss the two main approaches of this topic:

Distributed Computing approach: where processing power is distributed across multiple nodes. Each node might be responsible for processing a subset of the data or a specific part of the computation. The results from each node are then

combined to generate the final result. Distributed computing makes it possible to process larger amounts of data more quickly by leveraging the power of many computers.

Distributed Databases approach: This approach involves distributing data across multiple databases, which may be located in different physical locations. Each database operates independently of the others, but they can work together to process queries. Distributed databases can improve data reliability and availability because of distributing data in multiple separated databases and physical devices.

This kind of data streaming processing system divides the data into small pieces called "records," "events," or "messages," which are then processed in a pipeline-like fashion. These systems can handle remarkably high volumes of data because they are distributed over many computers, each of which processes a portion of the data.

Data Streaming Technologies

Data streaming technologies are software platforms and tools designed to capture, process, and analyze continuous streams of data in real-time or near-real-time. These technologies allow organizations to extract insights from data as soon as it is produced, enabling immediate decision-making and response. They support various operations such as data ingestion, storage, transformation, and analysis, and are used in scenarios where timely data processing is crucial, such as financial trading, medical devices, and IoT applications.

They are a cornerstone of modern data architectures, particularly in scenarios that demand real-time or near-real-time processing. These systems are designed to handle massive volumes of continuous data flows, often from a multitude of producers.

Examples of data sources include sensors in IoT devices, social media feeds, log files, transaction records in e-commerce, among others. The data from these sources is often unbounded, expandable, and infinite, necessitating technologies that can process this data as it arrives, rather than in batches.

Moreover, it is more than worth mentioning that these technologies not only ingest and process data in real-time but can also store this data, either temporarily or for long-term historical analysis. They often provide mechanisms

for fault-tolerance and scalability, ensuring that the system can handle increasing volumes of data and recover from any errors or failures.

One of the key aspects of these technologies is their ability to support complex event processing. This includes operations such as filtering, aggregating, and correlating data, allowing organizations to detect patterns, trends, and anomalies in their data as soon as it is produced or arrived.

Data streaming technologies are diverse and include systems like Apache Kafka, Apache Flink, and Amazon Kinesis. Each of these systems offers different features and capabilities, catering to different use cases and requirements. The choice of technology often depends on factors like the volume and velocity of data, the required latency of processing, and the specific processing operations needed.

Several commonly used data streaming technologies provide the infrastructure to handle large-scale, real-time data streams.

1. **Apache Kafka:** LinkedIn streaming tool, that is a highly scalable and fault-tolerant data streaming platform that provides a publish-subscribe model for handling real-time data feeds.
2. **Apache Flink:** Apache Flink is a powerful stream-processing framework that supports both batch and stream processing jobs.
3. **Apache Samza:** LinkedIn streaming technology, that is a distributed stream processing framework that uses Kafka for messaging and Apache Hadoop YARN for fault-tolerance, processor isolation, security, and resource management [1].
4. **Amazon Kinesis:** Provided by Amazon Web Services (AWS), it is for analyzing streaming data.
5. **Apache Storm:** is an open source distributed real-time computation system that can reliably process unbounded streams of data [2].
6. **Apache Spark Streaming:** Apache Spark Streaming is indeed a crucial technology in the realm of data streaming. It enables scalable, high-throughput, and fault-tolerant stream processing of live data streams [3]. Data can be ingested from many sources, then can be processed using complex algorithms expressed with high-level functions. One of the significant advantages of Spark Streaming is its integration with Apache Spark's core, allowing it to process both batch data and real-time data [4]. This is a critical feature for use cases that require a unified platform for processing real-time and historical data. Under the hood, Spark Streaming

receives the live stream of data, divides it into small batches, and processes these small batches of data using the Spark Engine. The processed results are then returned in batches, thus enabling real-time data processing.

Just a reminder, that each of these technologies has its own strengths and is better suited to certain use cases than others. The choice of technology will depend on the specific requirements, such as the nature of the data, the volume of the data that needs to process, and the processing requirements.

Practical:

To delve deeper into the intricacies of distributed data streaming processing, I conducted a methodical experiment (This work is inspired by “Staleness Control for Edge Data Analytics” [5]). Through this endeavor, I aimed to understand the dynamics of data handling, processing, and prediction-making, especially in the context of Electric Vehicles (EVs).

Objective:

My goal was to design a robust distributed data streaming system capable of real-time data management from multiple edge computing nodes. This system would make predictions regarding the need for an EV to charge at the next station, based on the data processed.

Data Generation:

Simulating the environment of 13 Electric Vehicles (EVs), I consistently ran scripts to ensure continuous data production. Each EV passed through 4 edge nodes during its journey.

Infrastructure & Mechanism:

Edge Nodes: As EVs passed through, nodes collected data, made predictions about charging needs, and forwarded this information, along with periodic model updates, to a central server for further analysis and aggregation.

Central Server: After receiving data and models from nodes, the central server aggregated and trained model (based on the received data), then sent back an updated global model to each node for better prediction accuracy.

Model Training: Nodes utilized the incoming data to continually refine their predictive models.

Flow:

As the EVs traversed the network, they encountered 4 edge nodes. These nodes not only received the data but also provided predictions about whether the EV should charge at the next station. The nodes relayed this data and their model predictions to a central server, which subsequently returned an aggregated global model to each node to enhance their prediction capabilities.

Experimentation:

I executed the experiment in two distinct phases:

Neural Network Model: Here, I tested two configurations on the central server:

- a. Training the model post-aggregation.
- b. Simply aggregating the models and dispatching the global model to nodes.

Linear Regression: The central server was initially set to train the aggregated model. However, an observation emerged: merely sending the aggregated model without additional central training proved more effective.

Observations:

Model Aggregation at Central Server: The accuracy of the global model directly correlated with the number of models aggregated:

Aggregating 4 models: High accuracy

Aggregating 3 models: Moderate accuracy

Aggregating 2 models: Poor accuracy

Model Freshness: Real-time, freshly updated models from nodes ensured the global model's high accuracy. However, as the number of nodes sending outdated models increased, the accuracy of the global model deteriorated:

All 4 nodes sent real-time models: Optimal accuracy

1 node sent an outdated model: Slight dip in accuracy

2 nodes sent outdated models: Noticeable accuracy degradation

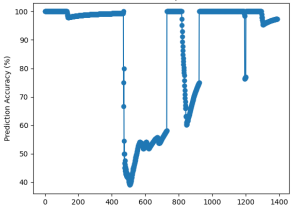
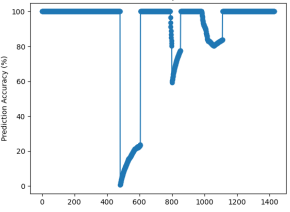
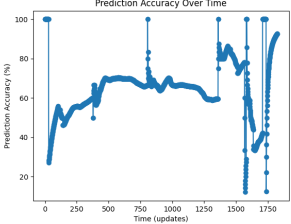
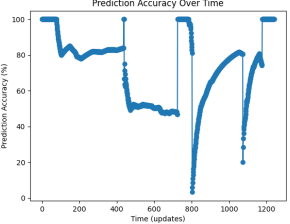
3 nodes provided outdated models: Poor accuracy

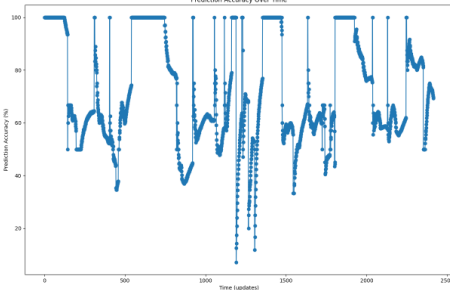
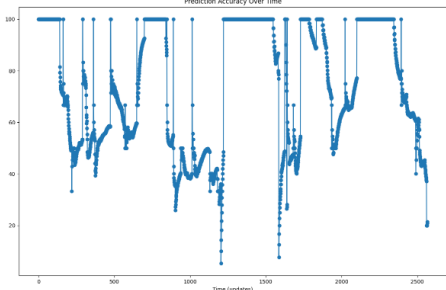
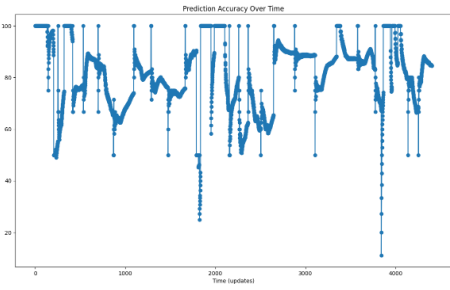
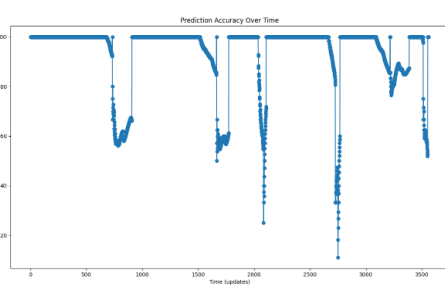
All nodes sent outdated models: Extremely poor accuracy

Results & Visualization:

A comprehensive table documented the outcomes of both experiments, highlighting the comparison between Neural Network and Linear Regression models. This was complemented with illustrative charts for a visual representation.

Car script	13 cars to produce data	<p>the script produces dataset and uses kafka as a producer</p> <p>Car Information: Car ID, Car Model - Current Status: Current Speed, Location, Node, Car Status - Battery Information: Battery Capacity, Current Charge Level, Battery Life - Distance and Consumption: Distance Covered, Distance to the Next Charging Point, Charge Consumed in the Current Step, Total Distance Covered - Environmental Conditions: Weather Condition, Traffic Condition, Gradient - Emergency Situation: Emergency Type, Emergency Duration - Timestamp: Simulated timestamp - Additional Information: Consumption Rate, Charging Stations, Battery Life Degradation Rate, Degradation Factor, Simulation Speed Factor, Real Start Time, Simulated Start Time, Time Step, Distance Step - Kafka Data Sending: The script sends data to Kafka topics, including 'electric_car_data', 'node1_data', 'node2_data', 'node3_data', and 'node4_data' based on the current node.</p>
Node script	4 nodes to store, train the model, and giving predictions	<p>continuously receives data from a Kafka topic, predicts whether a vehicle needs charging, updates a machine learning model with new data, and periodically exchanges the model with a server. It also monitors and plots the prediction accuracy over time. Moreover, it loads a pre-trained model and uses it in the beginning.</p>
ML models	Neural Network model and random forest	<p>Load Data, Data Cleaning: Adds a 'needs_charge' column based on a condition. Converts categorical variables into dummy/indicator variables. Removes duplicate rows. Removes non-numeric columns. Handles outliers by replacing extreme values with medians. Feature Selection: Selects specific features based on Recursive Feature Elimination (RFE) with a logistic regression model. Visualizes feature distributions and relationships. Data Visualization, Random Forest Classifier: Builds a Random Forest Classifier model. Performs hyperparameter tuning using GridSearchCV. Evaluates the model's accuracy and provides a classification report. Conducts cross-validation to assess model performance. Neural Network Model: Constructs a Neural Network model using TensorFlow/Keras. Compiles the model. Defines callbacks for early stopping and model checkpointing. Trains the model on the training data. Evaluates the model's performance on the test data. Prediction Function, Model Saving, Model Loading, Additional Predictions.</p> <p>This ML aims to cover various aspects of data preprocessing, feature selection, model training, evaluation, and prediction for a binary classification task related to determining whether a car needs charging based on its attributes and environmental conditions.</p>
Central Server	1 central server to receive the models from nodes, make refinement and aggregation, broadcast the general model to nodes	<p>A server that collects machine learning models and their accuracies from multiple nodes, aggregates these models using a simple average when a threshold is met, and sends the aggregated global model back to the nodes that request it. This setup can be used for federated learning or collaborative model training across multiple nodes.</p>

Nodes	Node - 1 -	Node - 2 -	Node - 3 -	Node - 4 -	 	
Cars	11 cars	12 cars	not more than 12 (depends on battery charging level)	1 certainly connected and 12 not certainly connected (depends on battery charging levels)		
Results from nodes	at starting 99%	at starting 99%	at starting 99%	at starting 99%		
Results if all models send fresh model	The general model becomes more accurate, specially, when it drops down by the time because of changing environment conditions	The general model becomes more accurate, specially, when it drops down by the time because of changing environment conditions	The general model becomes more accurate, specially, when it drops down by the time because of changing environment conditions	The general model becomes more accurate, specially, when it drops down by the time because of changing environment conditions		
Results if only three send nodes fresh models	The general model becomes more accurate, but less than all models.	The general model becomes more accurate, but less than all models.	The general model becomes more accurate, but less than all models.	The general model becomes more accurate, but less than all models.	 	
Results if only two nodes send fresh model	Not accurate	Not accurate	Not accurate	Not accurate		

Nodes	Node - 1 -	Node - 2 -	Node - 3 -	Node - 4 -	
Cars	11 cars	12 cars	not more than 12 (depends on battery charging level)	1 certainly connected and 12 not certainly connected (depends on battery charging levels)	
Results from nodes	at starting 100%	at starting 100%	at starting 100%	at starting 100%	
Results if all models send fresh model	The general model becomes more accurate, specially, when it drops down by the time because of changing environment conditions	The general model becomes more accurate, specially, when it drops down by the time because of changing environment conditions	The general model becomes more accurate, specially, when it drops down by the time because of changing environment conditions	The general model becomes more accurate, specially, when it drops down by the time because of changing environment conditions	
Results if only three send nodes fresh models	The general model becomes more accurate, but less than all models.	The general model becomes more accurate, but less than all models.	The general model becomes more accurate, but less than all models.	The general model becomes more accurate, but less than all models.	
Results if only two nodes send fresh model	Not accurate	Not accurate	Not accurate	Not accurate	
<div><div><div>Prediction Accuracy Over Time</div></div><div><div>Prediction Accuracy Over Time</div></div><div><div>Prediction Accuracy Over Time</div></div><div><div>Prediction Accuracy Over Time</div></div></div>					

Concluding Thoughts:

While the generated data was random, it showcased the potential variabilities in a real-world scenario. This experiment underscored the importance of fresh data and model aggregation strategies in optimizing prediction accuracy in distributed systems. Future endeavors might benefit from exploring real-world data to ascertain any variances in outcomes.

Experiment links:

<https://github.com/Maitham16/p1>

For more testing models and technologies regarding to this work:

<https://github.com/Maitham16/practical1>

Resources

- 1- <https://www.slideshare.net/HumbertoStreb/apache-samza-72469084>
- 2- <https://storm.apache.org/>
- 3- <https://hub.packtpub.com/spark-architecture-and-first-program/>
- 4- <https://spark.apache.org/docs/latest/streaming-programming-guide.html>
- 5- [https://eprints.cs.univie.ac.at/7072/1/Staleness Control for Edge Data Analytics Two Page .pdf](https://eprints.cs.univie.ac.at/7072/1/Staleness%20Control%20for%20Edge%20Data%20Analytics%20Two%20Page.pdf)