Distributed Contextual Anomaly Detection from Big Event Streams

BAKHTIAR AMEN

Thesis Supervisor: Prof. Grigoris Antoniou

Thesis Co-supervisor: Dr. Violeta Holmes

A thesis submitted to the University of Huddersfield in partial fulfilment of the requirements for the degree of *Doctor of Philosophy*

University of Huddersfield
School of Computing and Engineering
University of Huddersfield, Queensgate, Huddersfield, HD1 3DH, UK

January 2018

Abstract

The age of big digital data is emerged and the size of generating data is rapidly increasing in a millisecond through the Internet of Things (IoT) and Internet of Everything (IoE) objects. Specifically, most of today's available data are generated in a form of streams through different applications including sensor networks, bioinformatics, smart airport, smart highway traffic, smart home applications, e-commerce online shopping, and social media streams. In this context, processing and mining such high volume of data stream becomes one of the research priority concern and challenging tasks. On the one hand, processing high volumes of streaming data with low-latency response is a critical concern in most of the real-time application before the important information can be missed or disregarded. On the other hand, detecting *events* from data stream is becoming a new research challenging task since the existing traditional anomaly detection method is mainly focusing on; a) limited size of data, b) centralised detection with limited computing resource, and c) specific anomaly detection types of either *point* or *collective* rather than the *Contextual* behaviour of the data. Thus, detecting *Contextual events* from high sequence volume of data stream is one of the research concerns to be addressed in this thesis.

As the size of IoT data stream is scaled up to a high volume, it is impractical to propose existing processing data structure and anomaly detection method. This is due to the space, time and the complexity of the existing data processing model and learning algorithms. In this thesis, a novel distributed anomaly detection method and algorithm is proposed to detect *Contextual* behaviours from the sequence of bounded streams. Capturing event streams and partitioning them over several windows to control the high rate of event streams mainly base on, the proposed solution firstly. Secondly, by proposing a parallel and distributed algorithm to detect *Contextual* anomalous *event*. The experimental results are evaluated based on the algorithm's performances, processing low-latency response, and detecting *Contextual* anomalous behaviour accuracy rate from the *event* streams. Finally, to address scalability concerned of the *Contextual events*, appropriate computational metrics are proposed to measure and evaluate the processing latency of distributed method. The achieved result is evidenced distributed detection is effective in terms of learning from high volumes of streams in real-time.

Dedications and Acknowledgements

First and foremast, I would like to express my sincere gratitude to my advisor Professor Grigoris Antoniou for his unremitting support, guidance and patience during this Ph.D. research. He has been very supportive, encouraging, enthusiastic and intense adviser. I very much appreciate all his time, ideas and critical constructive feedbacks that immensely contributed to my professional skills. I would like to thank my second advisor Dr. Violeta Holmes for her support and valuable suggestions, including Dr. Violeta's research team members from High Performance Computing Group, for providing technical supports. I gratefully acknowledge the funding scholarship received from the University of Huddersfield.

I dedicate this thesis to my beloved mother, whom she has always been encouraging me to study further and made her dream come true. I also would like to dedicate this thesis to my beloved partner Mrs. Ban Rashid for her limitless love, encouragement, and support that I unremittingly received from her through this Ph.D. research. The progress of this work would not have been achievable without her, my siblings, mother and father-in-law. Last but not least, I would like to thank to all of my research colleagues at the University of Huddersfield, whom they have directly or indirectly supported me during this research; it has been a great experience to support each other and share skills with each other during the Ph.D. journey. Most importantly, I would like to thank Dr. Ilias Tachmazidis, and my friends Mr. Shwan Boskani and Mr. Sardasht Mahmood for their assistance throughout this research.

Table of Contents

List of Abbreviations

Abbreviations	Descriptions
A-SPOT	Adaptive Stream Projected Outlier
API	Application Programming Interface
BIRICH	Balanced Iterative Reducing and Clustering Using Hierarchies
BS	Bayesian Network
CA	Contextual Anomaly
CCA	Common Correlated Attribute
CE	Complex Event
CEP	Complex Event Processing
CQ	Continuos Query
DAG	Direct Acyclic Graph
DBMS	Database Management System
DDM	Drift Detection Method
DCAD	Distributed Contextual Anomaly Detection
DISD	Data Intensive Scientific Discover
DSMS	Data Stream Management System
DSP	Distributed Stream Processing
DSP	Data Stream Processing
DSPE	Distributed Stream Processing Engine
E3S	Event Stream Spout Splitter
EDDM	Early Drift Detection Methods
ETL	Extract Transform Load
FIFO	First-In First-Out
GMM	Gaussian Mixture Model
GPS	Global Positioning System
HMM	Hidden Markov Model
IBRL	Intel Berkeley Research Lab

ICA	Independent Component Analysis
IoT	Internet of Things
JVM	Java Virtual Machine
LOF	Local Outlier Factor
MAE	Mean Absolute Error
MP	Message Passing
NN	Neural Network
PCA	Principal Component Analysis
PHT	Page Hinckley Test
RAD	Real time Anomaly Detection
RMSE	Root Mean Square Error
SP	Stream Processing
SPE	Stream Processing Engine
SPIRIT	Streaming Pattern Discovery in Multiple Time-Series
SPOT	Stream Projected Outlier Detector
CESA	Contextual Event Stream Anomaly
SVD	Singular Value Decomposition
SVM	Support Vector Machine
TSA	Time Space and Accuracy
UI	**1.1.1** User Interface
WSN	Wireless Sensor Networks

Chapter 1

Introduction

The innovation of technologies and Internet connectivity are evidenced that this world is adapting from traditional to digital-based. Specifically, in the last decade, due to advanced technologies, high volumes of data sources from log records, call records, biomedical records, stock exchange, social media, network traffic, and manufacturing sensors are generating in different formats of (e.g., structured or unstructured). Thus, a new scientific paradigm has emerged under the umbrella of big data so-called Data Intensive Scientific Discover (DISD) (Chen et al., 2014, p.173). The term of "big data" is now universally used and became to a central for researchers and practitioner's attentions across multi-disciplines of such as bioinformatics, geophysics, astronomy, engineering, meteorology, e-commerce and social media. The literature of big data is very broad and there is not yet a formal definition from neither academia nor industry, however, Chen et al. (2014, p.173) and Tsai et al. (2015) defined it as;

"Datasets which could not be captured, managed, and processed by general computers within an acceptable scope".

In other words, the volume and velocity of generating data is beyond the capacity of current technologies to process, handle, and provide computational results. This is due to the limited computing resource, data

structured model, and complexity of the existing of the algorithms. Thus, this idea is motivated the industry and scientist to redesign computer hardware's (e.g., multi-core processor, cloud computing) based on the Moor's law to become more powerful than ever before (Tsai et al., 2015). Prior to the scalability concern, discovering hidden knowledge and predicting unusual *events* from high volumes of data is remaining to be a challenging task, specifically, from high volumes of data streams. In general, big data analytic comprises of data integrating, processing and analysing large-scale of both static and stream data formats. Thus, detecting unusual activities from big scale of data plays an important role in many application domains including air traffic monitoring system (Katal et al., 2013), network attack (Hashem et al., 2014), transaction frauds (Chen et al., 2014), weather broadcast, faulty sensors indicating oil and gas leakage (Xie, et al., 2011), and diagnosis from medical records (Ma, et al., 2016).

The main concept of anomaly is referring to unusual *events*, specious activities or different pattern in the dataset (Candela et al., 2009; Zhang, 2013). The study and literature of anomaly detection method is very extensive in information theory, machine learning, data mining, and statistics (Grosse & Turin, 2012; Gupta et al., 2014; Ma et al., 2016). Importantly, anomaly can be referred to positive or negative aspects in different application domains, for example, in network intruder detection, the network system administrator aims to trace suspicious activity from incoming traffic to make an immediate action against the intruder. In banking industry, detecting online frauds and suspicious activity is considered as one of the most priority concerns to protect client's account and funds. In network sensor domain, anomaly can be beneficial in detecting fault or error in the sensors. From the perspective of safety concerns, tracing and predicting incidents in real time is very important in many applications including highway road traffic monitoring system, airport surveillance, medical diagnosis, civil security, and engineering (Gupta et al., 2014). Similarly, detecting and predicting disaster like floods, storms,

and earthquakes are also major concerns in weather broadcast domain. In social media, for example, anomaly plays an important role in detecting user opinion behaviour from writing inappropriate comments (e.g., race and sexual abuse, arranging riot activities, online activities including terror and criminal threats).

1.1 Research Motivation

Internet of things (IoT) or Internet of Everything (IoE) are the two new emerged fields of the computer science. Today, creating high volumes of data is an easier process than it was in the previous decade; this is due to the low cost of IoT devices and other digital applications. For example, the size of connected object is expected to be one trillion sensors by 2030 (Yang & Fong, 2015); this includes 350 billion annual meter readings, power plants, machinery data, and Global Positioning System (GPS) (Yu & Lan, 2016). The main benefit of such trend is to provide consumers with affordable and secure energy supply (Zhang, 2013), while consumer and supplier could both have energy consumptions in real time and predicting extraordinary *events* and activities such as faulty sensors, energy leakage, or tampering meters. Importantly, the majority of data is generating in a form of stream by different applications and the size of these data is very large in scale.

On the one hand, detecting anomaly in real time plays a significant role in monitoring unusual behaviours from big digital devices such as; home suppliers, smart meters, smart motorways, smart city, work locations, and airport surveillance. Thus, online anomaly detection and mining from high volumes of data in real time is appeared to be a new research direction. The existing and traditional anomaly detection methods are mainly focusing on a specific type of point or collective anomaly problem in offline analysis. On the other hand, recent works of anomaly detection methods are mainly

disregards the scalability of the data concern; this is due to the proposed data structure model and computational complexity of the proposed algorithms. Considering processing and detecting anomalous *events* from 1 terabyte of data centrally (over a standalone machine), this possibly requires several hours or days to process and provide computational results with another major concern of network overloaded. Thus, detecting anomalous *event*s from high volumes of IoT sensor stream is an emerged research filed of big data stream mining (Duarte et al., 2016; De Francisci Morales, 2016; Bifet et al., 2016). The main motivation of this research is to develop a novel algorithm and method that will be able to detect *Contextual* behaviours of large sequence of IoT sensor data streams in real time. The following challenging tasks have been mainly studies and investigated in the thesis.

Online Learning: unusually online learning algorithms are required to process data in several subsets of streams in a sequence rather than process all the data at once. This is due to the need of real time processing structure and detecting anomalous *events* from streams are requires single-scan learning; once stream is processed, such data stream can be irrelevant and or it can be discarded at the later stage. Online learning is playing an important role in many dynamic monitoring applications such as network security, road traffic, healthcare diagnoses, airport traffic control, fire safety, and weather broadcast.

Scalability: as the size of data stream scales up, standalone machine is only capable to process and handle limited size of IoT data stream; this is due to limited memory space of the most proposed computing resources, dynamic evolving of streams over the time, and network bandwidth (Duarte et al., 2016; Schneider et al., 2016). Thus, in recent years, the concept of parallel and distributed approaches is increasingly attracting the attentions of both researchers and industry engineers to address the scalability problem. Importantly, most of the existing anomaly detection methods are designed to

detect anomaly centrally. Thus, research on anomaly detection over high volumes of data stream is limited; parallel partitioning and processing is required to compute several tasks at once with low-latency and real-time response (Gupta et al., 2014). However, to handle such high rate of data streams, robust parallel and distributed stream detection is suggested to be an alternative solution. The main benefits of parallel and distributed processing can be summarized in; a) high throughput *event* streams (1 million *events* per second) in real time, b) low-latency computational response which is very important for anomaly detection in real time, and c) overcoming computational resource constraints (Candela et al., 2009; Grosse & Turin, 2012; Amen & Lu, 2015).

Contextual Anomaly: selecting anomaly detection type is one of the key priorities challenging task in many big data application domains, specifically, in streaming application domains. Existing anomaly detection methods are mainly either focused on *Point* or *Collective* anomaly types, however, research on data stream *Contextual* anomaly detection is limited (Folino & Sabatino, 2016; Karunaratne et al., 2017). Thus, new *Contextual* anomaly detection from high sequence of data stream can be a challenging task. For example, consider highway road traffic scenario for speed monitoring of vehicles over consecutive time-series as depicts in figure 1.1, where blue vehicles are representing those vehicles within national speed limited of 120km/h and red vehicles are over speeded vehicles. Consider high volumes of vehicle speeds data coming from road traffic IoT sensor in unbound of sequence of streams in real-time. An important question can arose, what is the main appropriate anomaly detection type (*Point, Collective or Contextual*) for the stream data? This question will be answered in the next section with detailed description of each anomaly type.

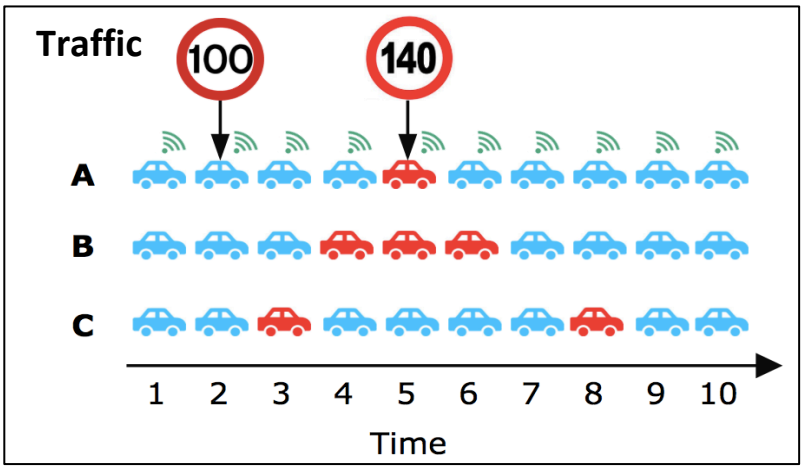


Figure 1.1: Anomaly detection types for the time-series data scenario in road traffic monitoring system.

In general, IoT devices are generating data in a form of streams; thus, data are arriving in a sequence of streams with time stamped on. Consequently, sequential analysis can be an appropriate solution to propose for data stream anomaly. Consider three types of anomaly detection in figure 1.1. Scenario A refers to an individual vehicle's speed behaviour within the sequence of the data streams, thus, a single *event* at t_5 is considered as *Point* anomaly. In this context, *Point* anomaly is considered as one of the most common approaches in many application domains, while collective anomaly refers to collection of unusual *events* from the data instances (set of points). A group of vehicles with over speed behaviours of 140km/h from t_4 to t_6 are considered as Collective anomaly in scenario B. In contract to these, *Contextual* anomaly is classified based on the relations between the data instances *Contextual* and behavioural attribute. The most important impact on *Contextual* anomaly is a time of *event* occurrence [26]. Scenario C can be considered as *Contextual* behaviour of the same vehicle at two different occasions t_3 and t_8, hence, the same vehicle's speed of 140km/h recorded with the same context at different time metrics. The benefit of *Contextual* anomaly is to define the behaviour of the *event* stream in a specific context. This is one of the most appropriate detection types to detect the contexts of data behaviour in the time-series

domain (Candela et al., 2009), (Duarte et al., 2016). *Contextual* anomaly can be defined based on the data instances and time occurrence of the attributes.

In summary, these challenges are primarily motivated this investigation, and according to the literature, existing research studies have disregards to investigates in-depth to the levels of distributed *Contextual* anomaly detection. The main goal of this study is to address and propose novel distributed *Contextual* anomalous *event* stream detection, specifically, detecting *Contextual* behaviours from large sequence of IoT sensor streams in parallel.

1.2 Research Questions

The primary aim of this thesis is to design a scalable *Contextual* anomaly detection approach in real time to handle high rate of *event* streams from IoT sensors. To achieve this aim, the following research questions are defined.

1. What are the main existing methods to detect anomalous *events* from sequence of IoT sensors in real time?
2. Is it possible to detect anomaly dynamically regardless of streams high rate and to what extent the proposed algorithm is capable to address and handle changes over the stream distribution without human interventions?
3. Can an algorithm detect the *Contextual* behaviour in the large sequence of data streams based on using window partitions stream data structure model?
4. On what scale detecting *Contextual* behaviours from high sequence volumes of IoT sensors in parallel is possible?
5. How is it possible the proposed *Contextual* stream behaviour detection method and algorithms to solve similar other real-time application problems?
6. What are the appropriate methods to evaluate the performance of both change detection and prediction error rates in the data stream?

1.3 Research Contribution

1. To address the research aim and objectives, the following significant contributions will be achieved. Studied and highlighted the potential problems of existing anomaly detection will be highlight and studied from distributed computing paradigm prospective.
2. This research identifies the *event* stream problem, defines and designs novel *Contextual Anomaly* $\boldsymbol{C_A}$ model to detect unusual *event* in the different context.
3. Designs novel window algorithms to partitioning high volumes of *event* streams into several *event* partitioning to protect *event* streams from changes and concept drift drawback.
4. Implements **Contextual Event Stream Anomaly (CESA)** algorithm to detect changes and *Contextual* behaviour from large sequence of IoT sensor stream based on DSPE data structure model.
5. Designs new **Distributed Contextual Anomaly Detection (DCAD) Framework** to address scalability data anomaly constraints with a comparison result of centralised and decentralised performance results.
6. Analyse and evaluated the experimental results for proposed algorithms based on several evaluation metrics.

In summary, this thesis flows from theoretical to experimental perspective. First, anomaly detection can be studied as a unique approach to detect anomalous *events* from IoT sensor stream in real time. This can be achieved by designing a new *Contextual* anomaly detection method based on the high number of scoring contexts in parallel per each window partitioning, since this approach is particularly absented in the existing solutions. Second, the proposed distributed method will be able to handle high throughput of *event* streams in real time with low processing time. Third, the evaluation metrics

will measure the accuracy of the proposed algorithm, which is based on the estimation of the scoring rate and algorithm's performances among the predicting error rates. The proposed algorithm and accuracy of the computational results are critically concerned to validate the algorithm performance. A detailed description and results of evaluation metrics are presented in (Section 5.6 and 5.7).

1.4 List of Publications

The main research contributions of this thesis are based on the following peer-reviewed publications. The publications are based on the theoretical and experimental study of big sensor stream anomaly detection in real time.

Paper I: Amen, B., & Lu, J., Sketch of Big Data Real Time Analytics Model, The Fifth International Conference on Advances in Information Mining and Management (IMMM), ISSN: 2326-9332, ISBN: 978-1-61208-415-2. Brussels, Belgium, June 21-26, 2015.

Paper II: Amen, B., Antoniou, G., Holmes, V., Tachmazidis, I. Distributed Contextual Event Stream Detection. 30th International Conference on Scientific and Statistical Database Management. Bolzano-Bozen, Italy, July 9 - 11, 2018.

Paper III: Amen, B., Antoniou, G.

A Theoretical Study of Anomaly Detection in Big Data Distributed Static and Stream Analytics . 20th International Conferences on High Performance Computing and Communications (HPCC), the (HPCC-2018), Exeter, UK, 28-30 June 2018.

Paper IV: Amen, B., Antoniou, G., An efficient Approach to Detect Big IoT Contextual Event Stream Anomaly Real time **(**2018**).**

1.5 Thesis Structure

This thesis is organised into five chapters as follows.

- **Chapter 2** covers research study literature for three domains of anomaly detection, stream mining and distributed data processing in parallel. First section discusses anomaly detection methods, second section describes anomaly detection in streaming domain including stream definition, and stream data processing structure model, and the third core section in this chapter is discussed the existing related works of anomaly detection methods in parallel.

- **Chapter 3** establishes theoretical foundation of the *event* streams problem definitions with proposed novel distributed *event* stream partitioning design methods. The distributed partitioning method is mainly based on the window partitioning technique with designed Contextual Event Stream Anomaly (CESA) algorithm.

- **Chapter 4** describes designed phases of anomaly detection framework of Distributed Contextual Anomaly Detection (DCAD) and its architecture to address two main research problem constraints of stream detection and scalability of high throughput *events* in real time.

- **Chapter 5** covers the experimental performed results and evaluation for the proposed algorithms based on two IoT case studies to estimate the accuracy, effectiveness, and scalability of proposed the DCAD.

- **Chapter 7** provides the thesis's conclusion with the summary of the problem, discussion of research limitations, contribution and implantations, including the experimental research results and overview of the future work opportunities

Chapter 2

Anomaly Detection: Background and Related Work

This chapter describes the research background and distributed anomaly detection research related works that are relevant to this thesis with focusing on three research domains; anomaly detection, data stream mining, distributed and parallel processing concepts. Section 2.1 describes an overview of traditional anomaly with driven characteristics to understand the concept of the problem. Since most of IoT data is arriving in a form of stream formats, detecting anomalous event from streaming data in real time is becoming a challenging task anomaly, thus, Section 2.2 describes the concepts of anomaly in data stream mining with stream notations, characteristics, model and processing techniques. Additionally, due to the lack of centralised based anomaly detection processing, decentralised and distributed is another research challenge to be concerned, about; Section 2.3 describes parallel and distributed computing methods in relation to high throughput (scalability) with low-latency and real time response challenging concerns. Section 2.4 discusses and covers anomaly detection related works in the area of parallel and distributed computing from this domain prospective; Information Theoretic-based, Statistical-based, Classification-based, Clustering-based, Density-based, Distance-based, and Online-based.

2.1. Anomaly Detection Overview

2.1.1. Anomaly Definitions

The term of "anomaly" is differed between one discipline to another, importantly, outlier, anomaly, and novelty terminology can be correlated, but in practice they are different. A formal definition of such concept is depending on the detection method in each application domain. For example, in statistical analysis, data is considered to be fitted into a normal model and outlier refers to those data which are distinct from the proposed model (Aggarwal, 2016), while the normal behaviour is based on predefined notion of normal objects in the dataset. Faria et al. (2013), Faria et al. (2016), and Schneider, Ertel et al. (2016) argues that both anomaly and outlier have the same definitions in terms of dissimilar pattern or anomalous behaviour in the data. Similarly, in data mining, outlier refers to anomalous pattern in the dataset compared to the remaining data (Zhang, 2013). Consequently, Beigi, Chang et al. (2011) explained that outlier possibly refers to a noise or irrelevant system behaviour, while noise could be due to network failure or reading measurement errors. Similarly, Aggarwal (2016) and Zhang (2013) argued that the difference between noise and outlier, and agreed on that noise is a weak type of outlier. In (Yang, Meratnia et al.. 2010) defined the noise as a potential source of outlier which possibly occurs due to faults in the sensors.

In network security, anomaly refers to intrusion detection, while such behaviour refers to fraud detection in financial sectors (Amen & Lu, 2015; Candela et al., 2009; Grosse & Turin, 2012). To conclude this, according to the literature definitions, it can be argued that outlier is more related to unusual pattern or behaviour in the static data; on the contrary, anomaly can be referred to an anomalous event in dynamic data (stream). Importantly, high score output results of anomaly rate are more achievable rather than the outlier result in the

majority of the applications, this is due to the clear understanding of anomaly objectives in each application domain as described in Section 2.1.

2.1.2. Anomaly Detection Characteristics

Anomaly detection from static data analysis is mainly can be learnt on offline as described in (Chandola, Banerjee et al., 2009). Similarly, the literature survey of outlier, novelty, change, and anomaly detections for temporal data (e.g., spatial-temporal data, data streams, time-series data, distributed data, and network data) are presented in (Chandola, Banerjee et al., 2012; OReilly et al., 2014 ; Yang, Meratnia et al., 2010; Zhang, 2013). According to these studies, anomaly detection for both static and streaming data is primarily based on a number of common facts as described in below.

i. **Data Domain:** one of the primary challenging task in anomaly detection is to define a data type in order to be able to provide answers to the problem during the data analysis or prediction. As described in Section 2.1, the nature of data type from one application to another is different, and data can be from collection of data instances (e.g. objects, events, records, vectors, patterns, observations), where every instance perhaps includes a number of attributes (categorical, binary or continuous). In addition to these, the input data can be univariate (single attribute) or multivariate (multi attributes).
ii. **Anomaly Type:** in general anomaly is categorised into three types of *Point*, *Collective* and *Contextual*. *Point* anomaly refers to an individual data instance behaviour (single point) compared to the rest of the other data instance behaviours. *Point* anomaly is one of the most common detection type in various applications such as in credit card fraud detection (Van Vlasselaer, 2015), weather forecast prediction (Erfani et al., 2016), network intrusion detection (García-Teodoro et al., 2009). The literature study of this type of anomaly is very broad, specifically,

in statistical data analysis, pattern recognition, machine learning, and data mining. These disciplines are mainly based on addressing classification and clustering problems (Beigi, Chang et al., 2011; Pham, Venkatesh et al., 2012). On the other hand, *Collective* anomaly refers to a collection of unusual *events* or behaviours from the data instances (set of points). However, these behaviours can be grouped into clusters based on similarly behaviours in unsupervised learning (e.g., machine learning). This can be achieved by using a number of techniques such as Markov Model to detect subsequence probability of the data and label the data instances as anomaly, and similarity distance metrics (Ma et al., 2016). For example, Ye and Li (2017) proposed *Collective* anomaly to detect unusual behaviours over the data streams with similar concept. Similar approach is used for sensor network detections by (Ma et al., 2016), for social network detections by (Akcora, et al., 2014; Ferrari & Kantarcioglu, 2014), and for multiple spatial temporal detections by (Zheng et al., 2015). Alternatively, Hidden Markov Model (HMM) is another appropriate model to detect subsequence probability of the data and label the data instances as anomaly (Zheng et al., 2015). *Collective* anomaly approach is proposed in various application domains to detect a group of dissimilar data behaviours including; Ye and Li (2017) proposed *Collective* method to detect a group of sensor network behaviours, this method is also advocated for social network behaviours in (Akcora et al., 2014; Ferrari, & Kantarcioglu, 2014), and for multiple spatial temporal detections in (Zheng et al., 2015).

Lastly, Contextual anomaly is associated with the relations between both data instance's Contextual and attributes since the most important impact on Contextual anomaly is the time of event occurrence (Gupta et al., 2014). For example, consider monitoring conference room normal temperature degree as depicted in figure 2.1, On the one hand, the room temperature is 26 °C at t1 when the room is occupied, while similar

temperature degree at t1 in midnight is considered as Contextual behaviour, this is due to the fact that similar behaviour occurs in the different contexts with different attribute value. In finance and banking industry, Contextual anomaly is associated with customer's spending behaviour, however, these limitations are based on the bank's spending credited threshold per account holder. For example, spending large amount of money (e.g., $1000) at Christmas is considered as normal behaviour, while similar spending behaviour in April is concerned as an unusual event (anomaly) (Van Vlasselaer, 2015).

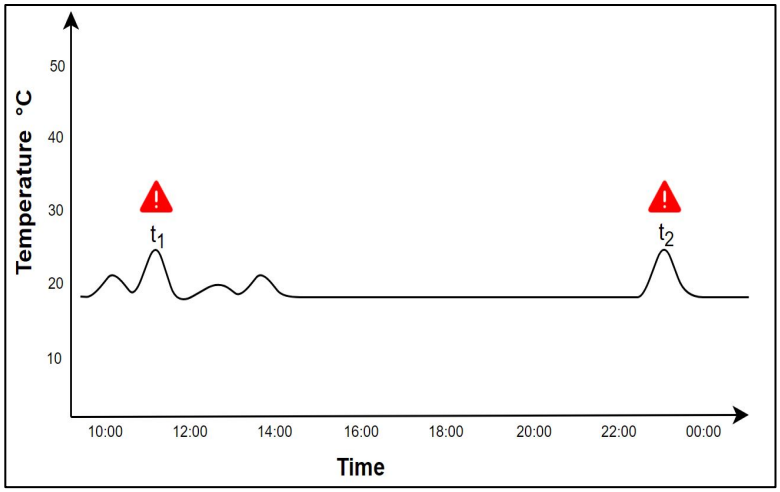


Figure 2.1: Contextual anomaly for the conference room temperature scenario.

Contextual anomaly is proposed to predict stock market shares (Golmohammadi & Zaiane, 2015), social networks behaviours between different group of users (Akcora et al., 2014), sensor network pattern detection (Hayes & Capretz, 2015), text data and semantic analysis (Mahapatra et al., 2012). Importantly, *Contextual* anomaly can also be used for online shopping customer's behaviours; for example, customer's shopping behaviours can be changed from one season to another within the similar spending range, but with different interest (context) (Jiang et al., 2014).

Anomaly detection types are combined methods (e.g. *Point* and *Contextual*, or *Collective* and *Contextual*) in a number of research studies to address and discover different research problems. For example, Mirsky et al. (2017) combined both *Point* and *Contextual* anomalies based on using pcStream algorithm to protect user's mobiles from malicious activities. Similarly, in (Hayes and Capretz, 2015) Point and *Contextual* anomalies have been combined to detect faults from high volume of sensor networks data. Yexi, J, (2014) proposed *Contextual* and Collective anomalies to detect unusual behaviours of computer clusters memory consumption behaviour.

iii. **Output Label**: the output of anomaly results is either based on label or score results. The result techniques are based on the proposed anomaly detection algorithms (e.g., supervised, unsupervised or semi-supervised), specifically during the learning process for training of a model (Faria et al., 2016), prior knowledge of the data behaviour is required to be known. A significant human effort is required to propose manual labelling or obtaining data labelling for the data training in some of the anomaly detection approaches. For example, consider labelling 1 millions of data instances manually is believed to be time consuming, complex, and very expensive procedure (Chandola, Banerjee et al., 2009). As data stream is changing over the time and labelling data stream is impractical in most real time situation. On the other hand, scoring output refers to the assigning an anomaly score, for example, to the sequence of data instances or to the window partitions and such approach is described in (Section 3.8). There is extensive literature and research of output scoring techniques over sequence of data in (Chandola, Banerjee et al., 2009; Chandola, Banerjee et al. 2012; Zhang, 2013).

In summary, the aforementioned of anomaly overview is mainly based on the statistical data analysis point of view, in contrast to anomaly detection in non-

stationary data where the data in form of stream is significantly different. In dynamic situation, anomaly is required to be detected in real time and the learning processing can be considered on online. A detailed description of data stream, stream models, stream processing, and anomaly detection in streaming are described in the next sections.

2.2. Anomaly Detection in Data Stream Analytics

2.2.1. Stream Definition

A formal definition of stream is described by Muthukrishnan (2005) as;

"a sequence of digitally encoded signals used to represent information in transmission".

Streams are generating at very high rate by diverse applications from IoT sensors, online transactions, traffic networks, stock market, online web clicks, medical records, manufacturing machines, and social media (Golmohammadi & Zaiane, 2015) .

2.2.2. Data Stream Model

Stream model is defined as logical formula of the stream data structure and stream computational model is one of the most common models to represents streaming data format (Erfani et al., 2016).

Definition 1: data stream S refers to stream with unbounded of items/elements, in contrast to static data; data streams are infinite and arrive at a very high rate. As denoted in Equation 2.1, S_1 refers to the first instance of the stream while each stream instance consists of *tuple* which compromises with a timestamp (e.g., (s_1, t_1))

$$\boldsymbol{S} = \{\boldsymbol{S_1}, \boldsymbol{S_2}, \boldsymbol{S_3}, \ldots,\} \tag{2.1}$$

Definition 2: Data stream is potentially infinite ($N \rightarrow \infty$) and completed data stream is impractical to be stored neither on memory nor disk due to the high rate and size of the streams. For such reason, data streams can be divided into *sub-streams* of *tuples* as denoted in Equation 2.2.

$$\boldsymbol{S_i} = \{(\boldsymbol{s_1}, \boldsymbol{t_1}), (\boldsymbol{s_2}, \boldsymbol{t_2}), \ldots. (\boldsymbol{s_n}, \boldsymbol{t_n})\} \tag{2.2}$$

Since $\boldsymbol{S_i}$ consists of an unbounded sequence of *tuples* (***s, t***) and ***s*** refers to individual data instance arrived at time ***t***. Accordingly, *tuple* is associated with either *implicit* or *explicit* timestamps *t*; while both categories are depending on the application underline assumption where the data stream is created from or arrive into the system. The *implicit* timestamp refers to arrive time of the *tuples* as they entered into the system (Tran, Gaber et al., 2014). However, the *implicit* timestamp can be added to an arriving *tuple* or if the timestamps are missing. In this context, such problem can be addressed with *time-based* windowing mechanism (See Section 3.4). On the contrary, *explicit* timestamp refers to the embedded timestamp to the data sources when the *tuple* is created by the real-world systems (Babcock et al., 2002). The *explicit* timestamp can be used to re-order of the data stream *tuples* into a sequence of ordered timestamps. The main disadvantage of *explicit* is correct ordering the timestamps from the transmission system; for example, *tuple* t_2 could possibly arrive before *tuple* t_1. A details comparison of timestamps detail is described in (Chaudhry, 2005).

In IoT applications, sensor data can be modelled and measured as a sequence of streams and they can be considered as time-series data. The reading value and time-series can be correlated in the sequential data analysis; thus, they can be modelled as *key-value* pairs of *tuple* (s_i, t_i). In the real-world applications, sequence data can be either discrete or continuous (time-series) (Chandola, Banerjee et al., 2012). In this situation, the IoT data stream is considered as continued data instance with timestamps. Aggarwal (2007), Amini (2013), Bifet (2009), Ma et al. (2016), and Muthukrishnan (2005) all agreed on the three fundamental requirements of data stream constraints in most of the application domains according of *Time, Space, and Accuracy* (TSA) metrics.

- **Requirement 1:** Data stream continuously arrives at a very high rate (e.g., millisecond). Thus, real time learning, and analysis is significantly important, while such requirement is impossible in offline learning.

- **Requirement 2:** Data stream generates in unbounded sequences of data

instances ($N \rightarrow \infty$). Storing potentially infinite data streams on memory is inappropriate. Thus, suggested solution is partitioning data streams into sub streams with single-scan over the data stream.

- **Requirement 3:** The nature of data stream changes over the time and change occurs in data stream sequence, thus, proposing an appropriate and novel computational method is a challenging task.

Furthermore, the problem of the data stream is broadly studied and investigated by many research communities in neural networks, machine learning, data stream mining, big data stream analytic, and social network analysis (Hu et al., 2014; Philip Chen & Zhang, 2014). The next section describes the data structure of stream model and stream formulations during the data stream processing and mining.

2.2.3. Anomaly Detection in Streaming Data

In dynamic situations, data streams can be generated by various applications and anomalous *events* possibly occur due to the result of either system behaviour (e.g., sensor) or changes in nature of the data distribution. Thus, sudden changes in the data records can be referred to the anomalous *event* and such behaviour is considered as an *event* detection (Aggarwal, 2016). On the other hand, in machine learning, specifically, in supervised learning, change is referring to a novelty detection, mainly when the classifier is missed such behaviour within the training process. In recent years, several studies have investigated novelty detection problems based on offline and online approaches for the multi-class label of data streams (Faria et al., 2016; Krawczyk et al., 2017). In addition to this, change possibly occurs in several conditions such as during data transformation, grouping data clusters, feature disappearing, class label swaps, float probability distribution or data discards. Importantly, Gaber et al., in (Tran, Gaber et al., 2014) descried Change detection as:

"Change detection is the process of identifying differences in the state of an object or phenomenon by observing it at different times or different locations in space."

On the other hand, gradual change in the data stream value and trend can be related to the concept drift, this is based designed model with prior unknown environment while according to Aggarwal (2016) change is not considered as an anomaly and concept drift defined as in follows:

"Concept drift refers to a change in the class definitions over time or underlying class (concept) of the data changing over time".

A detailed review of concept drift with taxonomy of concept drift detection methods in data streams is described in (Gama et al., 2014; Kuncheva, 2008). In general, concept drift detection refers to the problem of supervised classification learning scenario (Farid, Zhang et al., 2013). The proposed model first designed based on prior knowledge of the data behaviour in advanced. For example, the concept of the underline data stream at time t must be to the same of the newly arriving data stream at $t+1$, in contrast, the assumption output is considered to the concept drift problem (sudden change). The detection behaviour is mainly depending on prior known of a use of a model based on learning estimated training of the data samples. Consider an example of network intrusion detection learning supervised algorithm (e.g., classification learning) based on the decision tree structured design. The model is designed based on human prediction of expertise to construct the model tree according the estimated of all sudden change (e.g., suspicious activity) within arrived traffic data streams.

Alternatively, the estimation can be considered according to the data distribution behaviours in unsupervised behaviour as described in definition 3. In general, proposed formal model can only be appropriate when a prior knowledge of application objective behaviour is known, then the assumption of detection model can be beneficial, however, these learning processed is mainly

referred to data mining and machine learning techniques.

Definition 3: Assume two stream sets (S_1, S_2) are observed with given two Probability distribution of (P_1,P_2) (Tao & Ozsu, 2009). The similarity of their estimated distribution is based on their computed distance (e.g., Euclidian) of $dist\ (S_1, S_2) = dist\ (P_1, P_2)$, as denoted in the following Equation 2.3.

$$\boldsymbol{dist(P_1, P_2)} = \sqrt{\sum_{i}^{n} (\boldsymbol{p_1(v_i)} - (\boldsymbol{p_2(v_i)})^2} \qquad (2.3)$$

Where *v(s)* is the value of the data in both stream assumptions (S_1, S_2), this is mainly based on a prior knowledge of the formal model construction. A probability of each $vi \in v(s)$ in S_1 and S_2 is based on the distance distribution in $p_1(v_i)$ and $p_2(v_i)$, if the probability between $P_1 \neq P_2$ is large, it assumes that the distribution S is changed. However, assumption of prior knowledge of the environment in many applications such as IoT data stream is always unknown due to the nature of the data distribution and dynamic behaviours of the sensor devices in real-time.

Furthermore, Gama (2013) characterised five types of change in data streams as (e.g., sudden (A), incremental (B), gradual (C), recurring (D), and outlier (E)) in figure 2.2. Consider five scenarios for the aforementioned concept of change types; as in on online shopping, for example, customers interest behaviour on a particular item can changes suddenly compared to their past interests, such shift can be considered as sudden (Type A). In retailer industry, loyalty card has a significant positive impact of the retailer's investment over a time; such progress change within the data refers to (Type B). The moment when the UK Brexit result is announced, the news data stream over social media, specifically, Twitter stream comments have become very popular, suddenly after several months such news deliberately becomes to less important, between since and now, Brexit news suddenly shifts and becomes popular again mainly when new Brexit legislation formally is introduced. Thus,

such behaviours are considered as a gradual change (Type C). A particular item sale can be very popular for a specific period (e.g., for a month or a year), and this item becomes less interesting to be purchased due to the market computation, thus, such behaviour is considered as (Type D). Similarly, property buyer's interest is changed from time to time between in each season. Lastly, in banking industry transaction fraud can be considered an outlier/anomaly against single account holder (Type E).

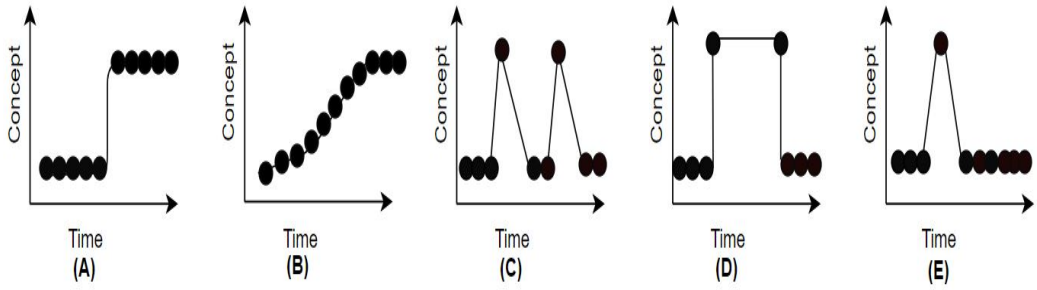


Figure 2.3: Concept drift detection types.

In the last decade, several machine learning methods have been used to detect change during the data stream distribution, for example, Ensemble classifiers (Farid, Zhang et al., 2013) and Drift Detection Method (DDM) (Gama et al., 2004), and Early Drift Detection Methods (EDDM) (Bifet et al., 2006). In Farid, Zhang et al. (2013) and Kuncheva (2008) ensemble (multi-classifier) method for both labelled and unlabelled data stream is proposed based on window sizes and using a threshold parameter for addressing both concept drift and change detection problems. Similarly, Kmieciak, and Stefanowski (2011) proposed supervised learning approach based on constructing a decision tree classifier to monitor probability distribution of a sudden change within the data streams. A similar research in Yang and Fong (2015) presented single tree learning classifier to detect concept drift detection within the data streams; this learning technique is mainly depends on the behaviour of the tree classification.

According to Gama et al. (2014), in the data stream mining, changes possibly occur mainly within online learning, specifically, in supervised learning, when the relation between data instance input and object is found to be different. A literature of change detection within data stream is provided by (Farid, Zhang et al., 2013) and (Joao Gama, 2013) with a taxonomy of detection methods for each (e.g. sequential contextual, control charts, and monitoring two distributions) data analysis. Furthermore, the main difference between each concept is that the former change detection refers to a labelled data (supervised learning), while the later detection relates to both situations of labelled (supervised learning) and unlabelled data (unsupervised learning) (Tran, Gaber et al., 2014). However, the computational complexity of the former learning approach is higher than the labelled data, due to the availability of both labelled data.

Overall, these methods are appropriate techniques to detect change from the data streams, while the main drawback of such methods are their capabilities with specific data stream type, limited size of data streams, and their data processing structured model since most of the existing detection methods are designed to process and detect data centrally. Therefore, detecting change from data stream in distributed and parallel computing can be an ideal solution to overcome scalability concern and handle high throughput of the data. Specifically, such approach can be managed with high levels of data throughput and real time response. The aforementioned methods mainly focus on either detecting changes in the nature of data distribution state in the learning process, or model behaviour regardless of the scalability concerns of sensor streams, medical streams, weather broadcast streams, network sensor streams. However, data processing is a major concern, specifically, for processing high volumes of data streams, thus, in recent years, many data stream; management systems and distributed processing is developed to offer continue queries with limited data source capabilities as described in the following sections.

2.2.4. Data Stream Collection Concept

Data stream can be collected through messaging system based on many-to-many communication service. This is emerged to help collect and transfer streams from many sources of (e.g., IoT sensors, web streams, and network streams). A messaging system is primarily based on two models: *point-to-point* and *publish-subscribe* (Tatbul, 2010). The former approach refers to direct single *point* of messaging communication mechanism as depicted in figure 2.3. The disadvantage of such approach is that only one message at the time from the queue can be delivered to the specified destination. Thus, this approach is incapable of, for example, delivering high volumes of data streams. Thus, high throughput and scalability are critical concern in this approach due to the system input and output communication data structured (Duarte et al., 2016; Schneider et al., 2016).

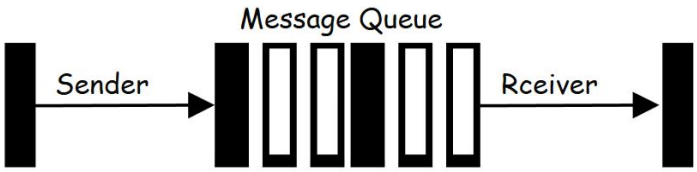


Figure 2.3: Point-to-point messaging system.

The other approach of *publish-subscribe* of messaging delivery is an alternative solution, and the communication service can be made through distribution of multi brokers as depicted in figure 2.4. *Publish-subscribe* is also known as producer and consumer; this approach is designed to deliver high volumes of streams in parallel decentralised (Jacobsen, 2005). The advantage of the distributed *publish-subscribe* system is the ability to integrate and delivery multi-sources, flexibility in (*pull-based, push-based*), high-throughput of data streams, and low latency communication response (Cugola and Margara, 2012). The main architecture of distributed *publish-subscribe* message consists of three components.

- ***Publishers:*** The main task of the publisher is to produce related streams to a subscriber in an asynchronous manner.

- ***Brokers:*** A broker assigns streams into e.g., *topic-based* and *content-based* partitions as shown in figure 2.4. The benefit of the broker is to filter irrelevant *events*; this helps to reduce the network bandwidth in each node and publish only requested interested streams to the subscribers.

- ***Subscribers:*** A subscriber receives those messages from publishers based on requested interested stream partition from *topic name* (e.g., *Temperature*).

In general, *publish-subscribe* messaging system is mainly comprised of *topic-based* or *content-based* systems. The main role of the *topic-based* system is to assign messages to a *topic*, where every *topic* is associated with the stream or *event topic* names. This approach is connecting messages from producers to the consumers based on the *topics* scheme as shown in figure 2.4. Filtering is one of the main drawbacks in this approach, for this reason, it can be a critical problem when the size of stream is scaled up, or when all streams are published on to the given *topic*. Alternative solution is to emit only forwarded request topic names to the subscriber.

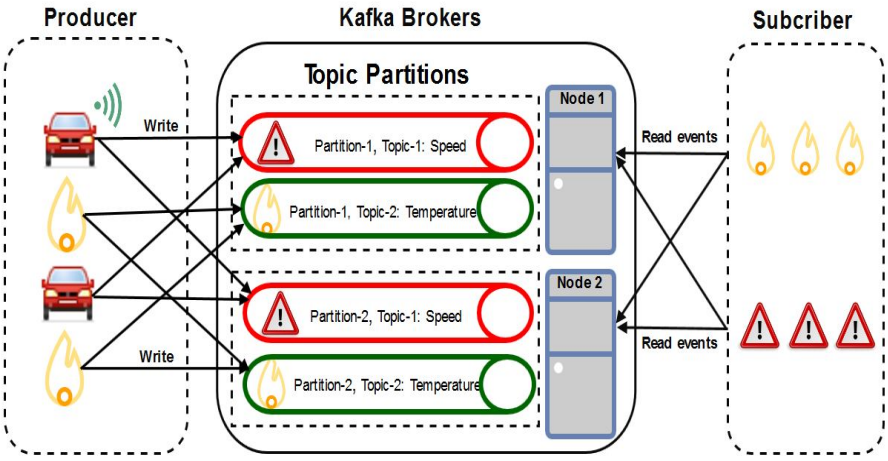


Figure 2.4: Distributed stream collection architecture in Kafka.

On the other hand, *topic-based* is proposed in many types of research problems e.g., social media *topic* detection, crowded scenes feature *topic* detection (Ye & Li, 2017), and linked stream data *topic* detection (Saleh et al., 2015). In contrast to the previous approach, the *content-based* approach is more flexible and it provides filtering function for each published stream (Plale, 2003), for example, the consumer can only receive stream that has been filtered according to the request from publisher. As another example, in a road traffic monitoring situation, subscriber can register a query to receive all vehicles with over speed *events* from national speed limited of e.g., 120km/h[1] as denoted in 2.4.

$$\{\text{type} = \text{vehicle}, \text{speed} > 120\} \tag{2.4}$$

The main advantage of such approach is that the flow of *events* to the subscriber is motivated by *event* content instead of predefined groups or *topics*. Thus, in the *content-based* approach, *events* can be filtered, and only interesting *events* can be forwarded to the subscriber for processing and computations, hence, this approach is also decreases overhead messages and handles a load balance on each node as shown in figure 2.4.

Eugster et al. (2003) argues that *publish-subscribe* system can support loosely coupled communication for a scalable system while, loosely coupled can be evaluated based on three dimensions of, time, space, and flow. Time decoupling is associated with the information of communication between *publishers* and *subscribers.* Specifically, publisher can produces new messages to subscriber even when the subscriber is disconnected; then, the data can be delivered when the subscriber is recovered or reactivated. Such characteristic is known as a dynamic and flexible communication in *publish-subscribe* messaging systems. A prior knowledge of communication and identifications of both *publishers* and *subscribers* are unknown, hence, such approach relates

[1] http://www.metric.org.uk/speed-limits Worldwide Highway Speed Limited Matrix

to space decoupling. Lastly, synchronisation decoupling refers to the interconnection communication between both *publish-subscribe* while such communication is considered to be synchronised to the consumer.

Overall, there are several enterprise *publisher-subscriber* messaging systems available including IBM MQ[2], Java Message System JSM[3], Active MQ[4], and Rabbit MQ[5]. These frameworks each have limitations and drawbacks in terms of scalability in handling overloads of streams with overhead network bandwidth, fault-tolerance, distributed architecture support, and guarantee in delivering high volume of infinite data streams in real time. A detailed survey of the most common and reliable distributed *publish-subscribe* messaging system is described in Kreps (2011) and Tatbul (2010).

Alternative solution of distributed *publish—subscribe* messaging system is proposed and developed by LinkedIn so-called Apache Kafka[6]. Kafka provides anonymous many-to-many streaming messaging service delivery (Philip Chen & Zhang, 2014). Apache Kafka[7] is a scalable distributed messaging system framework, which provides anonymous streams messaging delivery service in real time and guarantees high throughput and low-latency of streams delivery. Additionally, in the case of failure, Kafka has a replication strategy to replace the node tasks with the other node in the cluster and guarantees its messages delivery service. Kafka consists of three main components, *producers, topics*, and *consumers*. The main task of each component and implementation is described below.

Kafka also provides distributed messaging system approach with many features that other system is incapable to provide, such as *topic* partitioning, high throughput messages, and low-latency response. In recent years, Kafka has

[2] https://www.ibm.com/support/knowledgecenter/en/SSFKSJ_8.0.0/com.ibm.mq.pro.doc/q004870_.htm
[3] https://docs.oracle.com/javaee/6/tutorial/doc/bnceh.html
[4] http://activemq.apache.org/
[5] https://www.rabbitmq.com/
[6] https://kafka.apache.org/
[7] https://kafka.apache.org/intro

been widely proposed in many research problems to help with collecting and aggregating streams in real time; for example, Esposito, Ficco et al. (2015) proposed Kafka to aggregate data streams for the purpose of ontology extraction. Similarly, Kreps (2011) proposed distributed *publish-subscribe* framework to collect data logs with low latency performance in real time. Accordingly, the main benefits of the *publish-subscribe* messaging system are high-throughput and low latency (Kreps, 2011). In this context, *publish-subscribe* paradigm is an appropriate and reliable messaging system to be proposed in this thesis to aggregate large-scale of IoT sensor streams in real time as described in (Section 2.2.4).

2.2.5. Data Stream Management System (DSMS)

Aggregating data stream in real time is a key requirement of data processing. Data Stream Management System (DSMS) is one of the most common techniques to handle dynamic data in form of continues data stream. As data stream is emitted into the DSMS, it manages Continues Query (CQ) processing over the data streams to address the velocity problem. DSMS is capable to handle, process and retrieve data streams in real time for only limited size of data similar to the Database Management System (DBMS). A key challenging task in stream mining is to detect anomalous *event* from continues data stream and to manage high volume of data streams. Thus, the main purpose of stream management is to combine the stream data into appropriate format before to extracting any knowledge from them. DSMS offers a reliable and flexible mechanism to combine and store streaming data locally and provides Continues Query (CQ) over the arrived streams as can be seen in figure 2.5.

On the one hand, the advantage of CQ is that it can facilitate, handle, and organise such high rate of continues data stream. On the other hand, the disadvantage of DSMS is that, when the data size is scaled up, CQ can only

capable to process limited data centrally. Thus, detecting anomalous *event* throughout this technique is impractical due to the scalability of the stream. Alternative solution is the reduction technique, which technique also can be inappropriate, while streaming data is correlated, and *event* query is also mainly has a temporal condition. Time plays an important role in *event* detection, thus, conducting directly operator CQ on such high rate of data stream could result in either workload, difficult in complexity of computational result, hence, anomalous *event* possibly disappears or becomes disregarded. On the other hand, DSMS mainly adapts inherent timestamps to order data instances at the boundary in the Stream Processing Engine (SPE) and such timestamp can be disappear during the processing (Cugola and Margara, 2012). In recent years, new data stream structure model, this so-called *Data Stream Processing* (DSP), this has emerged to address such aforementioned drawbacks and provide low-latency response. A detailed description of such approach is presented in (Section 2.2.1).

In the last decade, many data stream management systems are developed to handle the scalability and other stream characteristics such as Extract-Transform-Load (ETL), INFOMIX (Genesereth, Keller et al., 1997), Aurora (Abadi et al., 2003), STREAM (Arasu et al., 2004), and TelegraphCQ (Chandrasekaran, Cooper et al., 2003). The main drawbacks of such DSMS frameworks are their data structure model and computational resource limitation to deal with big data characteristics and the lack of supporting distributed stream processing data structure model.

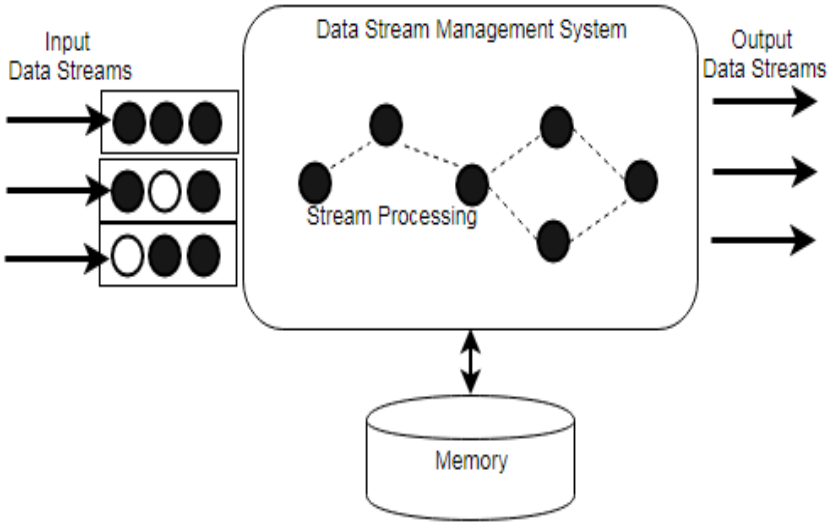


Figure 2.7: Data stream management system architecture.

.

2.3. Distributed and Parallel Data Processing

The concept of parallelism is generic; however, in recent years, due to the advances in technologies, the size of data has grown rapidly, and parallel distributed methods have been proposed to address the scalability of high volumes of the datasets. The main concept of distributed computing is to interconnect several computers and make communications through Message Passing (MP) to perform different tasks (Agarwal, Tayal et al., 2009). For example, figure 2.6 shows centralised and distributed data processing and mining approaches. The centralised approach (one the left) is associated with standalone machine for stream processing and mining data that are coming from IoT applications including weather broadcast and traffic monitoring system. While the distributed approach (on the right) refers to distributed stream processing and mining across number of computer nodes in parallel. Importantly, one of the most important aspects of distributed computing is a parallel execution to split large complex tasks and data into a

smaller sub-tasks to handle and produce computational results (Rauber and Rünger, 2013).

The concept of programming model in parallelism is associated with dynamic data partitioning across computer nodes. In general, parallel and distributed computing is a combination of parallel programming model (e.g., MapReduce) and computer application framework (e.g., Apache Hadoop[8], and Spark)[9] to perform distributed tasks and process high volume of datasets over different commodity architecture of either computer cluster or cloud computing (Esposito, Ficco et al., 2015). A term of parallelism refers to a dynamic partitioning of the continuous query over the input of the dataset based on one of the common programming methods (e.g., data and task parallelisms).

Data parallelism: relates to the data partition mechanism, where datasets can be divided into across of computer nodes. Map Reduce is one of the most common types of data parallelism to partitions and computes high volume of the dataset into a sub-set and partitioning them across different computer nodes in parallel. Map Reduce is based on the input data in a batch format (static format) in an offline mode and the process can be finished when the analyse task is completed; in contrast, *event* streams are arriving continuously at a very high rate where Map Reduce is incapable to handle such requirement of the stream.

Task parallelism: refers to the process of execution tasks made by different operators.

[8] http://hadoop.apache.org/
[9] https://spark.apache.org/

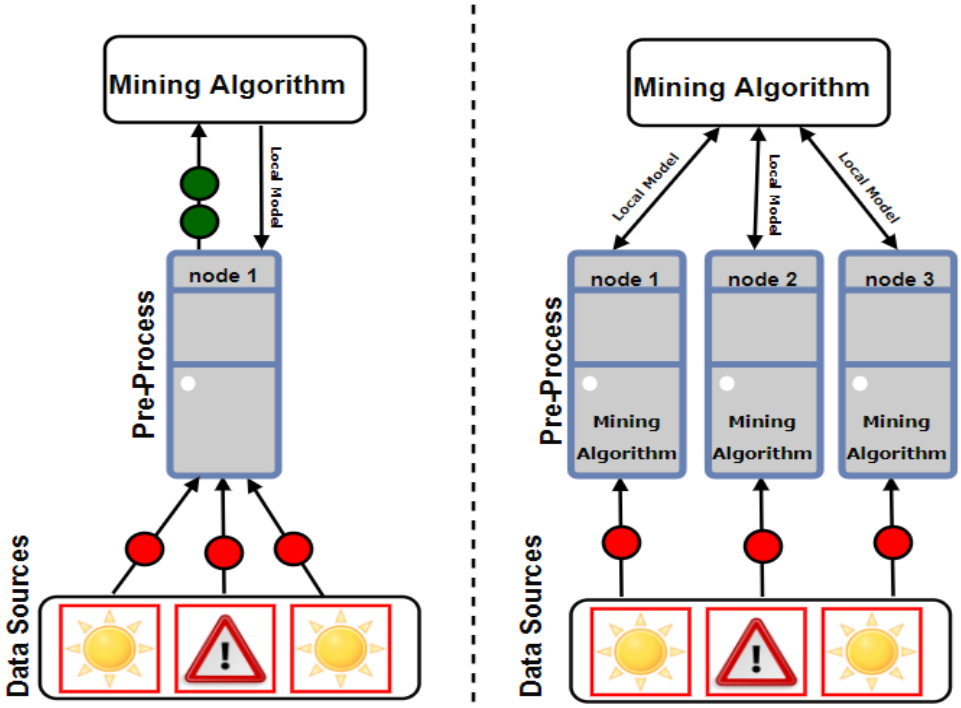


Figure 2.4: Centralised (left) and distributed data stream processing (right).

Big data is primarily based on two distributed data processing of *batch* (offline learning and *stream* (online learning) *analytics* (Philip Chen & Zhang, 2014). On the one hand, *batch analytic* is introduced to address the first (Volume) and second (Variety) characteristics of big data for large-scale of static data through offline learning. For example, many distributed storage systems such as HDFS, Cassandra, HBase, Hive, and GFS frameworks have been developed to run on Hadoop, and the aim of such frameworks are to address storage limitations of centralised databases and to run computational operations on. In addition to this, the *batch* approach is based on collecting; storing and analysing static data, and anomaly detection can be implemented over stored the static data regardless of considering the low-latency execution time and online learning (stream detection in real time). On the contrary, detecting potential *events* from streaming data requires online learning process and real time prediction, this is due to the nature of the stream characteristics and constraints as described in 2.2.2 (Karunaratne et al., 2017; Tran et al., 2014).

Since big data phenomenon is emerged in 2012, many researchers have been attempting to detect anomalies from large-scale of datasets including (Mohiuddin Solaimani, 2014; Wang, Shen et al., 2015; Yan, Zhang et al., 2015).

Theoretically, in batch analytic approach, anomaly detection refers to predicting the number of outliers from the static data with multi-scan learning approach over the datasets. In contrast to streaming data, anomaly refers to *event*, which occurs in real time, and it requires to be detected according to the same speed of the data stream as described in Sections 2.1, 2.2, and 2.3. On the other hand, *stream analytic* has emerged to process high volumes of data streams in real time with low-latency response and online learning prediction. Such approach is primarily based on Distributed Stream Processing (DSP) computational model to address big data three characteristics of (Volume), (Variety), and (Velocity). The concept of DSP depends on the dynamic stream partitioning, while all the partitioning mechanism is mainly based on two parallelism models; *data* or *task.*

In distributed and parallel processing, fault-tolerance or disruption during the learning execution in real time is a highly critical concern to guarantee processing high throughput streams in any DSP. For such reason, a number of *Distributed Stream Processing Engines* (DSPE) including Apache S4[10], Flink[11], and Storm[12] have been developed to address the aforementioned potential problems with similar stream processing data models. A detailed comparison of these framework studies is available in Appendix 1.

[10] http://incubator.apache.org/projects/s4.html
[11] https://flink.apache.org/index.html
[12] http://storm.apache.org/index.html

2.3.1. Distributed Batch Data Processing

Pervious section is introduced a distributed data processing concept for collecting and processing large-scale of datasets and streaming data. This section describes the most common distributed batch data analytic framework.

Apache Hadoop

Apache Hadoop is a distributed high throughput of batch data processing engine based on Map Reduce programming models. MapReduce is one of most reliable parallel programming model to analyse large-scale of the dataset (Philip Chen & Zhang, 2014). The concept of MapReduce is based on two common functions of *maps* and *reduces*. *Map* function sorts the datasets and shuffles them over computer nodes in order to find a similar matched pair from the data values, while *reduc*ing function is grouping the data values with the same attribute values in parallel as shown in figure 2.7. On the one hand, the main drawback of MapReduce is the re-execution processing and learning tasks. The iteration process of computational result is possible, thus, Map Reduce suffers from processing streaming data due to the constraints as described in Section 2.1.2. On the other hand, MapReduce has been proposed in many researches to address offline complex problems across different scientific area including in bioinformatics (MapReBio3), genetic data (MRscie1) engineering, and IoT (Hayes & Capretz, 2015; Zhang et al., 2016), environmental data. Map-Reduce is proposed by (Ma, Wu et al., 2015; Yan, Zhang et al., 2015) for sketching problem on Hadoop cluster for the large-scale of datasets. Additionally, Map Reduce also operated and deployed on different distributed computing architectures such as cloud computing, high-performance computing (Karatepe & Zeydan, 2014), and grid computing (Bai, Wang et al., 2016).

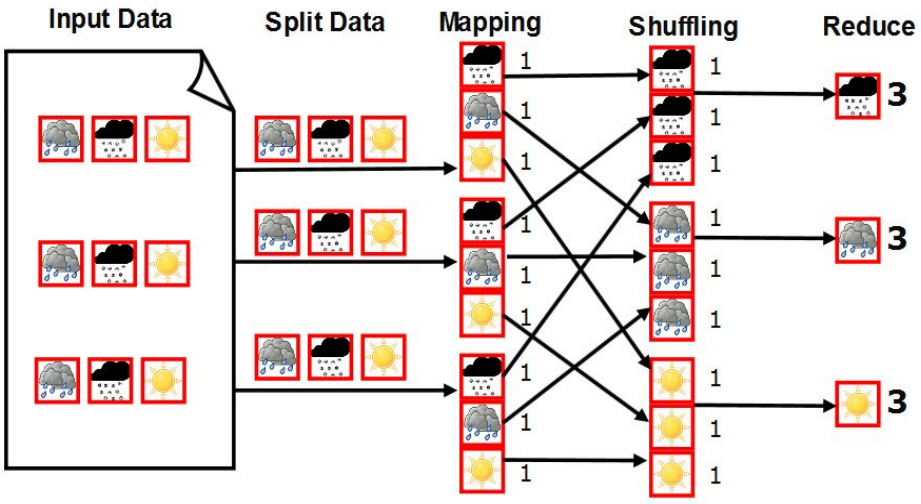


Figure 2.5: Map Reduce distributed programming model.

2.3.2. Distributed Stream Processing (DSP)

To evaluate the appropriate technique and method to detect anomalous *event* over high volumes of data streams, a theoretical background behind stream and distributed processing is required. Thus, understanding the concept of distributed stream processing data structural model is required for the sensor stream integration and pre-processing modules in Chapter 4 and 5.

Apache Storm

Apache Storm is a real time distributed stream processing framework with the capability of processing one million stream *tuples* per second on a standalone computer node (Storm, 2016). Similar to the Hadoop's MapReduce programming data model, Storm's programming model is based on three components of *spout, bolt,* and *topology* as shown in figure 2.8. *Spout* is known as a first entry point of Storm framework and the main

tasks of each *Spout* is to read and convert data stream into a *tuple* data format from messaging queue system like Kafka and Twitter API. A *tuple* is a pair of ordered values in a form of <*tuple, timestamp*> data format. *Bolt* is known as a computational unit of input streams; hence, *bolt's* computational functions are comprised of *filter*, *join*, *aggregate*, and *communicates* operations to execute different tasks including read and write to the database. The most important components of Storm's are *topology*; a *topology* can be viewed as graphical representation of stream programming model linking operation units to each other through streams. The structure of *topology* in Storm is made from *spouts* and *bolts* based on Direct Acyclic Graph (DAG) node representation. As can be seen in figure 2.8, Storm topology consists of *Spout* (Sp_1 and Sp_2) with (B_1 to B_5) *bolts*, through DAG made of stream connection (e.g., red arrows).

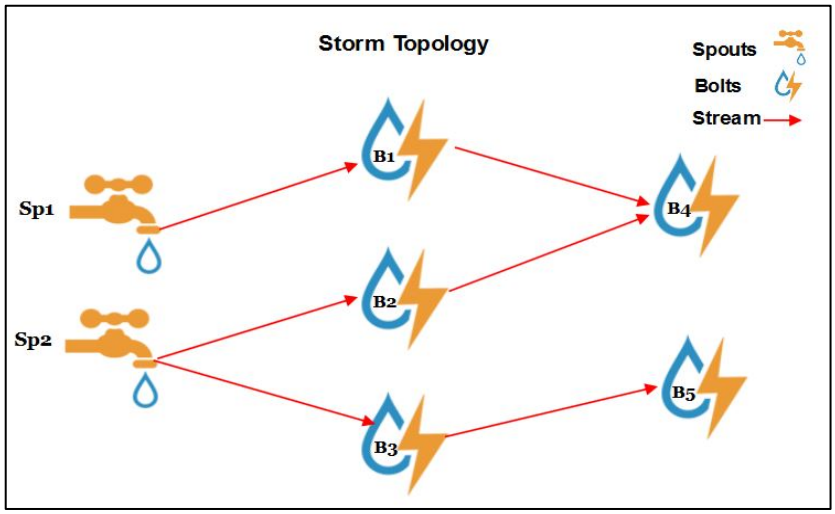


Figure 2.6: Apache storm topology programming model.

The *topology* builder defines the *topology* structure, *spout* and *bolt* from two streams, e.g., (*stream1*) and (*stream2*). The data stream then can be shuffled and grouped them over the different computer nodes based on *ShuffleGrouping*

mechanism. This type of grouping mechanism is one of the Storm's streams partitioning approach as described in the next section.

Storm has been implemented in many real time stream processing solutions including for the Twitter streams (Akter & Wamba, 2016), weather stream (McCreadie et al., 2013), IoT sensor streams (Kamburugamuve et al., 2015), and Social Media streams (Gao et al., 2015). Similarly, to batch data analytic, Storm has had a mayjor contribution in addressing many anomaly detection reseach problems, for examples, in Hu et al. (2014), Storm is proposed to detect anomaly from CPU data stream behaviour. This research is more related to the unusual behaviour of machines rather than solving a particular stream problem regardless of the data scalability concern. Other research in (Gao et al., 2015) attempted to implements distributed stream processing on cloud architecture to analyse social media streams. Such approach is mainly attempted to analyse social media through clustering algorithm, and the assumption of dynamic change in the data stream is disregarded, when there is unclear process of data stream partitioning tasks. However, Candela in (Candela et al., 2009, and Candela et al., 2012) argues that clustering is an unappropriate approach to detect anomaly from large data streams due to the fact that clustering tasks are more related to dividing data into a number of clusters rather than the data behaviour.

The architecture of storm is based on distributed infrastructure, which is made from *Nimbus*, *Supervisor* and *Zookeeper*[13] clusters as shown in figure 2.9 While *Nimbus* represents as a master at node on the top of the architecture with four *Supervisor* nodes and the connection between *Nimbus* and *Supervisor* is made by *Zookeeper* cluster, which is acting as coordinator. Processing latency between each storm component is playing an important role, since number of workers in each node is depending on the

[13] https://zookeeper.apache.org/

compatibility of each used computer node in terms of processing latency and memory space.

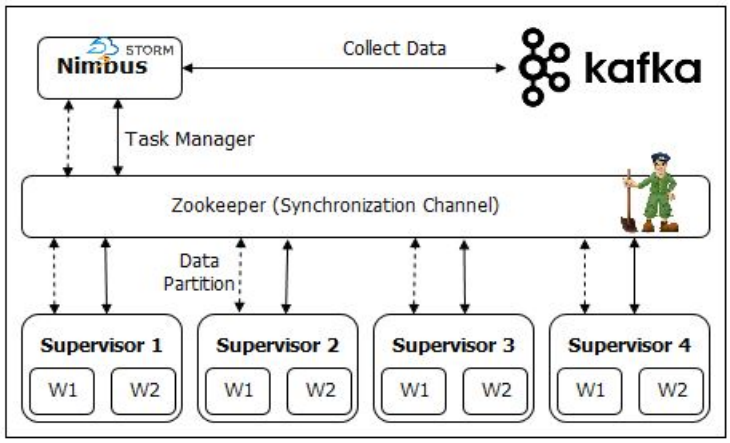


Figure 2.7: Apache storm architecture.

A detailed description of Storm architecture components can be described as follows.

i. ***Nimbus:*** *Nimbus* acts as Hadoop's master architecture, and the main task of Nimbus is to divide created topology's script codes across each computer nodes known as *Supervisors*. *Nimbus* assigns and manages computational functional tasks, which can be performed by each supervisor.

ii. ***Supervisor:*** A supervisor is known as a *slave* in Hadoop cluster architecture. It manages Storm's workers and the main task of *Supervisor* is to execute logical functions based on assigned tasks by the *Nimbus,* and to listen to the *Zookeeper* to excuse tasks from the workers. A worker in *Supervisor* also refers to Java Virtual Machine (JVM) and with constructed threats, which defines the tasks. Each worker comprises a number of executors and *tasks* while each *task* process data streams are implemented in *spouts* and *bolts*.

iii. ***Zookeeper:*** Apache *Zookeeper* is a high performance distributed coordinator, which maintains and monitors the health status of Storm cluster and acknowledge received messages. *Zookeeper* offers distributed data synchronisation mechanism, which is a critical concern in distributed computing environment (Philip Chen & Zhang, 2014).

The main concept behind high volumes of stream processing is a stream partitioning scheme in the DSP, hence, the aim of partitioning scheme is to define how the data stream can be processed or to be partitioned in parallel. In DSPE and framework such as Storm, partitioning task can be constructed from number operators (e.g., *bolt*) to process and emit data streams into the predefined destination. In this context, Storm offers various partitioning mechanisms and the most four common grouping techniques as described in below;

Shuffle Grouping: Data stream tasks can be shuffled randomly based on round robin scheme similar to Map Reduce data structure shuffles. Figure 2.10 illustrates the number of stream *events*, which can be partitioned across the number of workers while each work has been assigned to processes, e.g., an *event*. Streams can be processed equally, and the benefit of such approach is load balancing to prevent network overhead.

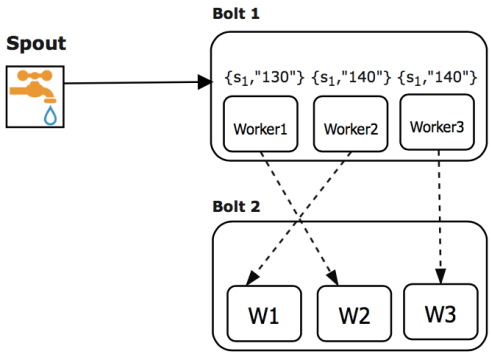


Figure 2.8: Stream partitioning shuffle grouping mechanism.

Filed Grouping: Streams can be controlled and grouped according to their data value in each of their schema and stream *tuple* values. For example, a similar value of *tuples* stream can be grouped in a jointed worker, for example, vehicle speed value tuple {s_1, "140"} can be grouped and joined by the same *worker2* in *bolt2* as illustrated in figure 2.11.

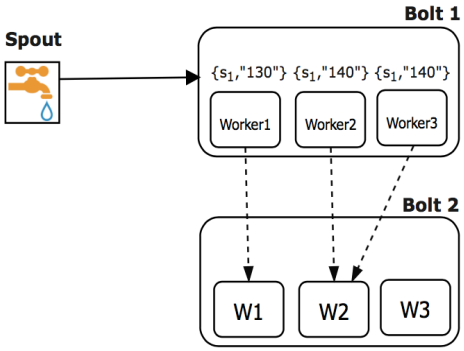


Figure 2.9: Stream partitioning filed grouping mechanism.

All Grouping: Copy of data stream *tuples* can be replicated to all the other *bolts* without partitioning them across different *bolts* as shown in figure 2.12. The disadvantage of this approach is overloaded data streams in each *bolt*. A key benefit of all grouping is that *events* stream tuple values can be all grouped by a specific e.g., *Bolt* 1.

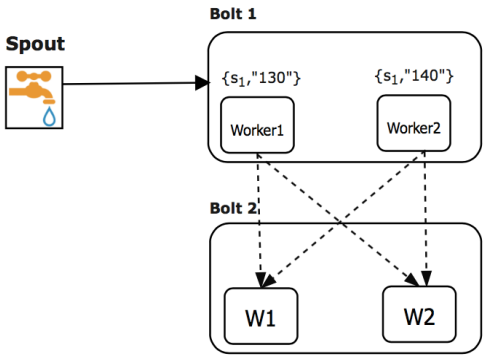


Figure 2.10: Stream partitioning all grouping mechanism.

Global Grouping: Global grouping is associated with joining all the data stream *tuples* from other workers in Bolt 1 into an individual worker (e.g.,W2) into *bolt2*. Computational results have been grouped them into a specific work within e.g., Bolt2 as shown figure 2.13. This can be achieved by defining an ID of each worker in every bolt within each supervisor node. For example, all the *events* can from *worker*1 to *worker*3 can be combined into *worker*2 in *bollt*2. This supports the redirecting tasks in the storm *topology* and synchronisation between each worker, since the drawback of this technique is that overhead of memory in each node is highly possible.

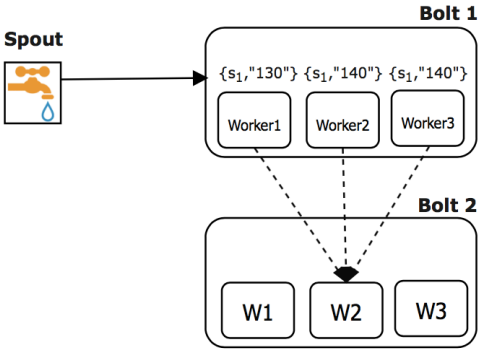


Figure 2.11: Stream partitioning global grouping mechanism.

2.4. The Complexity of the Data Stream and Size

Since IoT data streams are generating rapidly in a form of streams and due to the complexity of streams data structure, their data size can be very large and noisy; thus, it is impractical to detect anomalous *events* through current anomaly detection methods. Existing researches have mainly focused on addressing anomaly problems through using dimensionality reductions such as e.g., Sketches, Singular Value Decomposition (SVD) (Yan, Zhang et al., 2015), Principal Component Analysis (PCA), and Independent Component Analysis (ICA) (Muthukrishnan, 2005). The main drawback of dimensionality reduction is that in some situations such as in time-series, data attributes and objects are correlated with each other; for instance, temperature transmits several data values (e.g., high or low). Similarly, in monitoring real-life applications such as oil and gas leakage, fraud, and fire detections, data are generating in real time, thus, decision making is highly recommended before, for example, *event* can be irrelevant or dismissed. Thus, such decision making requires a robust data processing and online learning method. The main drawback of reduction techniques is that when the size of the data dynamically scales up, reduction techniques are possibly leads to a missing some of the critical *events* or stream tuple values can be missed. Therefore, in the process due to the high speed of the data stream (Chakrabarti, Keogh et al., 2002). There is a survey of outlier detection with low dimensional and high dimensional data reductions described by (Zhang, 2013).

In practice, summarisation technique is an alternative solution to decrease the data load and protect data from being lost. For example, Papadimitriou, Sun et al. (2005) proposed Streaming Pattern Discovery in multiple Time-series (SPIRIT) approach to summarise large collection of data streams. SPIRIT uses less memory, and this approach is focused on data correlations to prevent missing values from the high volume of data streams. SPIRIT approach is also adaptable to detect both sudden and gradual changes within the data streams

and to forecast an outlier. The main drawback of SPRIT is a data structured design for centralised-based approach. In Parthasarathy, Ghoting et al. (2007) argued that in centralised-based mining is incapable to handle high volume of data streams, specifically, the computational result can take a very long process, when real time computations and response are the main priority concern in most streaming application domains. In the last decade, alternative solution is proposed by (Erfani et al., 2016), to divide data streams into subsets of streams (chunk/portion) and across distributed nodes to handle such constraints as described in the previous section. On the other hand, data streams are generating in real time or near to real time and arrive at very high rate. Thus, data distribution changes over the time and monitoring newly arrived data streams and predicting their behaviours in real time is a challenging task.

2.5. Distributed Anomaly Detection Related Works

This section describes related anomaly detection methods, which have been proposed in parallel and distributed computing.

2.5.1. Information Theoretic-Based Anomaly Detection Method

Anomaly detection in Information theoretic method refers to the information content and observes with an impact of anomalies probabilities of according to the different measures (Chandola, Banerjee et al., 2012). In (Wu and Wang, 2013) a new concept of weighted complet entropy based on data distribution and attribute correlation proposed to measure the possibility of the anomaly candidate in large-scale of categorical data. Rettig et al. (2015) proposed another two information theoretic measures (Relative Entropy and Pearson Correlation) to detect large-scale of cellular network data behaviours by implementing such approach in parallel on Apache Spark. In this work, a

gradual change based on the Relative Entropy measurement first is detected. Then, Pearson Correlation and correlation metric have been conducted to detect abrupt changes in the data.

In summary, information theoretic can be measured based on the entropy method, this is more applicable for measures the approximation of categorical or spatial data format rather than streaming data. For such reasons, these measures are inappropriate for the streaming data, while the selection of such measures mainly depends on the numbers of anomalies in the dataset.

2.5.2. **Statistical-Based Anomaly Detection Method**

The study of statistical anomaly detection method is broad, a detailed description of such approaches is studied in (Chandola, Banerjee et al., 2009). The statistical technique is mainly referring to the assumption of the probability of normal model behaviour (training set) to determine if tested data fit into the normal model or not. In general, the statistical anomaly detection approach is based on parametric or non-parametric models and such approaches are proposed by (Rettig et al., 2015; Young et al., 2014) to detect the network intrusion behaviours. Summaries of both approaches are described in the followings.

In terms of parametric model, given dataset D generated from distribution D (θ) with unknown parameter θ, while θ can be estimated from available D to find $d \in D$. In this context, Gaussian distribution is one of the most common types of parameterised model in statistical-based method; example models are including Regression Model, Bayesian Network (BS), Hidden Markov Model (HMM), Gaussian Mixture Model (GMM). Consider a hypothesis of GMM for the observation of X when this value can be generated by an infinite number of Gaussian distributions. Every Gaussian Density $N(X|\pi k, \sum k)$ is a module of a mixture noted by mean πk, and covariance matrix $\sum k$. The computation of P

($\theta k \mid x$) initially defines and this value can be constructed from data sample x based on Bayes Rule of probability P ($\theta k \mid x$) as computed in Equitation 3.1.

$$P(\theta k|x) = \frac{\pi_k P(x|\theta_k)}{P(x)} \quad (2.5)$$

Where πk can be a mixing coefficient of the module k, which computed based on the probability of θk within x. Furthermore, in (Huang & Kasiviswanathan, 2015), autoregressive HMM is proposed to detect an unusual *event* in the data, however, in (Rettig et al., 2015) argues that HMM execution time is very demandable for training high volume of datasets due to the scalability size of the dataset which is inpactical for the model to be fitted. In contrast, prior knowledge of the data distribution in nonparametric is unknown. For example, data with a stationary probability distribution P can be estimated from given dataset D, while new data pointed x can be a new parameter and the relational assumption can be approximate. There two possible solutions available to be proposed to estimate the P based on the D or to decide if x is a random sample from P. Nonparametric model includes Histogram and Kernel-based approaches (Schneider et al., 2016; Su et al., 2007). Su in (Su et al., 2007), Schneider in (Schneider et al., 2016), and Huang and Kasiviswanathan (Huang & Kasiviswanathan, 2015) modified and optimized some of the nonparametric method to discover abnormal behaviours of the data and measured the proposed model based on the distribution of fixed data and micro-clusters. Candela et al. (2009) argues that nonparametric Kernel-based techniques are primarily capable when the assumption of the data generated from prior known distribution; however, this technique is possibly complex and inappropriate for the high dimensional volume of data streams. Detecting anomaly from data streams without known prior knowledge of the data structure based on nonparametric approach is described in (Beigi et al., 2011). The proposed data stream model is limited and incapable to detect changes within the data streams. Thus, the adaptability of the model is very critical in situation like

weather prediction, while the model is required to incremental the learning process to detect the change and return validate results. Similarly, in e-commerce and online shopping recommendation items, the proposed model is required to consider user's purchasing interest behaviour and the model must validates such unexpected change in the data distribution.

Aggarwal (2016) argues that the statistical methods computationally can be accurate, while both parametric and nonparametric methods are impractical to analyse large-scale of dataset. This is due to the validation results between theorises and computational as major drawback in data mining. In situation of anomaly detection, for example, labeling anomaly output manually may require human expertise and time considering validating the proposed model. To conclude, statistical models are incapable for online learning from dynamic data and learning from streaming data is more related to an online learning process.

2.5.3. **Classification-Based Anomaly Detection Method**

Classification method refers to supervised learning in machine learning and anomaly detection technique is mainly based on training anomaly model to test the output result of detection behaviours based on two learning assumptions of normal and anomalous labels (Aggarwal 2007). Data label availability is a major concern in supervised learning, as in some situation like streaming application the data label is unknown. In the last decade to address such problem, several classification models are proposed including Support Vector Machine (SVM), SVM refers to one-class label classification model-based, the learning process is based on divides the data into two sets of learning and testing. For example, in (Perkins 2003) detected novel behaviours from data streams based on one-class SVM classification. Similarly, OReilly et al. (2013) proposed one-class SVM technique to reduce a computational complexity of data sensors and detect outliers within each local node. As argued in

(Schneider, Ertel et al., 2016), OC-SVM is incapable to assign large-scale of labels for the model to learn and to detect anomaly due to the learning and predicting anomaly result process by the model. Alternative solution is proposed based on multi-class learning by (Hoens, Polikar et al., 2012) to address such problem to training data from multi labelled normal classes.

The literature of anomaly detection in classification-based method is extensively investigated. The proposed algorithms are categorised into tree-based algorithms includes (e.g., bagging and boosting decision tree, random forest, C4.5 decision tree and boosted stump), rule-based, Support Vector Machine (SVM), and Neural Network (NN) (Chandola, Banerjee et al., 2009). One of the most common proposed classification algorithm is decision tree. The algorithm is easily interpreted data into a tree-based learning procedure, this is based on hierarchical partitioning and each partition within the tree acts as independent node. The tree procedure is based on a common assumption of top-down approach learning where the tree develops from the root to the top.

2.5.4. **Clustering-Based Anomaly Detection Method**

Clustering-based method is one of alternative powerful meta-learning technique to analyse high volumes of data created by advanced applications. Clustering methods are referring to unsupervised learning. A taxonomy of the Clustering-based algorithms are described in (Amini et al., 2014; Yang & Fong, 2015) and (Fahad, Alshatri et al., 2014). These studies are categorised Clustering based on *partitioning methods, hierarchical methods, density-based methods, grid-based methods*, *and model-based methods.* In recent years, clustering methods are widely studied and proposed in data stream mining including to address problems across different application domains such as micro-blogging (Lee & Chien, 2013), web analytics (Facca & Lanzi, 2005).

In relation to the scalability concern, research on parallel and distributed clustering algorithm in the literature is limited, specifically, for clustering data streams. In (Zhang et al., 1997) proposed distributed clustering algorithm so-called Balanced Iterative Reducing and Clustering using Hierarchies (BIRICH). The main data structure for this algorithm is based on *CF* concept and CF-tree method to summarise the data streams into *CF* data structure. BRITCH splits leaf node of CF-tree and any *CF* vector with low density is considered as outlier or anomaly. According to (Silva, Faria et al., 2013), proposed data structure for storing the summary of the data stream is crucial to handle memory and space constraints. While *CF* is constructed from d-dimensional data point in the cluster. Splitting cluster $\{\underset{X}{\rightarrow}\}$ is based on i = 1,2,3, ..., N, and *CF* vector of the cluster, while the splitting criterion is mainly depending on data structure triple of CF according to cluster measurements from: centroid, radius, and diameter. This s based on according to the number of data objects that are represents by *N*, liner sum of the data instance *LS*, with the sum of squared data instance by *SS*.

The concept of *CF* is proposed in another distributed clustering algorithm so-called *DenStream* by (Charu C. Aggarwal 2003). *DenStream* is a density-based algorithm for clustering data stream, similar to BRITCH, *DenStream* proposes *CF* data structure with two additional *p-microclusters* and *o-microclusters* parameters. The algorithm is constructed based on, T_p *DenStream* and checks for *p-microclusters* to find a possible outlier *o-microclusters.* A detailed description of the *DenStream* algorithm extension is proposed by (Feng Cao 2006).

Another extension of *CF* structure is *Clustream* algorithm, it the data structure is based on two concepts of (online and offline) approaches. First, a statistical summary of the data stream is stored on member and maintained by microclusters, and then the input summary of data as captured on the online phase can be trained and tested on offline. The proposed algorithm computes maximum microcluster boundary based on the standard deviation of mean distance from the cluster centroid according to the factor *f*. As a consequence,

for every new data stream instance, two nearest microclusters can be merged based on their Euclidean distance measurement and each microcluster is required to be stored from time to time.

In summary, according to Chandola, Banerjee et al. (2009), clustering-based method is mainly appropriate to organise data into group of data instances instead of finding or detecting anomalies. For example, in dynamic application scenarios, it is inpractical to large-scale of store data stream and then analysis the data on offline. Thus, such assumption is argued in (Erfani et al., 2016), as less accurate computational assumption for stream data. For example, both *DenStream* and *Clustream* distributed clustering algorithms are mainly based on *CF* data structure; hence, these approaches are involving a data reduction. While the main drawbacks of detecting anomaly from data stream is dimensionality reduction. In (Schneider, Ertel et al., 2016) argued that one of the disadvantage of clustering is controls of outlier score when the threshold scoring range is defined, and the distance of k nearest neighborare becomes very complex.

In (Liang Su 2007) distributed data stream outlier detection is proposed from kernel density estimation technique based on dived-and-conquer method to partitioning the data streams into micro-clusters. Similarly, another approach of anomaly detection from data stream without prior knowledge of the data is proposed by (Beigi, Chang et al., 2011). Similarly, Yu and Lan (2016) proposed unsupervised anomaly detection technique based on matrix sketching of summarising the data streams to monitor the proposed stream model behaviour. According to (De Mencagli, 2016) sketching approach based on Turnstile model, and such a model is an inappropriate model for time-series data. In (Zhang, 2013) argues that increasing number of attributes in sketching is complex $O(N^2)$ in terms of both space and time constraints in the summarisation technique.

Alternative approach of distributed anomaly detection from large dataset based on density technique is proposed by (Wang et al., 2015), and the main concept of such approach influenced by data portioning grid-based method. A complete

dataset is divided into *d*-dimensional space grids within master-slave architecture, and distributed Local Outlier Factor (LOF) algorithm is implemented locally on each node to estimate the density of each data *tuples*. In (Zhang, 2013) argued that there is a lack of theoretical and practical capabilities of LOF's to discover and detect change in the data stream, specifically, during the dimensionality reduction of the data. Another anomaly detection method has been proposed by (Li Yu, 2016: Schneider, Ertel et al., 2016), the detection method is mainly based on similarity-based technique, it focuses on similarity of the test data based on similarity-based technique from the training data. The main drawback of similarity-based is online learning during in dynamic stream detection (Chandola, Banerjee et al., 2012). Alternative solution of data stream anomaly detection is proposed by (Zhang, Li et al., 2015), this approach is mainly based on Stream Projected Outlier detector (SPOT). Another extension of SPOT algorithm is Adaptive (A-SPOT) approach in (Zhang, Li et al., 2015). However, the main there is a research limitation of ASPOT in terms of both theoretical, for example, anomaly type, or definition of anomaly on online learning, and technique limitation in terms of, e.g., data partition and detection strategy point of views.

Chapter Summary

This chapter is described a global understanding of anomaly detection, specifically, anomaly detection in streaming application including describing the relations and distinguish between anomaly over static data and streaming data. According to Aggarwal (2016), several factors can significantly can influence the results of anomaly detection as described in Section 2.1.2. Thus, the main difference between previous related works and this thesis are: (a) existing methods mainly focusing on capturing only individual streams from e.g., IoT data sensor rather than multi-sensor streams while most of the existing

anomaly methods have disregarded the main concept of stream (change in nature of the stream); (b) this thesis aims to offer a novel *Contextual* anomaly detecting method in the data stream domain, while *Contextual* anomaly detection research method for the data stream is limited compared to the other two most common researches of *Point* or *Collective* anomaly types. Interestingly, the existing anomaly detection studies, and researches are mainly focused on individual stream behaviour, rather than data stream's context, specifically in the IoT applications, data is correlated and capturing *Contextual* behaviour is a new research challenging task. However, IoT data attributes are correlated and it can be beneficial to detect the *Contextual* behaviour; of the data rather than single behaviour, and (c) parallel anomaly detection is one of the most promising methods to overcome the scalability problems and low-latency computational response, specifically low detection computational results, while such requirements have been disregarded in most of centralised methods. In addition to these, distributed sensor network anomaly detection is becoming an interesting research study to investigate and detect distributed sensor behaviours. Some of these approaches may possibly achieve high detecting performance based on proposing distributed stream processing architecture, specifically, using big data state-of-the-art methods. Data partitioning, algorithmic structure and change detection are major concern and high demandable and ambiguous in some of these studies. Nonetheless, this suggests that these methods are simultaneously satisfying some of the requirements of outlier or anomaly detection regardless of anomalous *event* detection over a large-scale of data streams in real time. In this concept, some of these studies and related works have suggested that stream constraints, high throughput, and low-latency computational results are major concern in detecting high volumes of anomaly detection and they are required to be considered during the data streaming mining and anomaly detection methods. Since last decade, several machine learning and data mining algorithms have been developed to address problems of anomaly detection through proposed of offline learning methods, while these algorithms have mainly been designed to

learning from the model behaviour and depends on the data reduction techniques before the learning concepts applied to the data on the transit.

Chapter 3

Distributed Contextual Event Stream Problem Definitions and Designs

In this chapter, we describe our definitions of *event* streams, *Contextual* anomalous and the proposed novel model designed. A general stream definitions and model notation is described in Section 3.1. Section 3.2 describes *event* stream problem definitions and notations. Section 3.3 highlights the process of high volumes of stream based on the stream structure model along with window modelling concepts to handle and capture infinite sequences of large-scale of *events* in real time. Section 3.4 proposes novel designs of *event* stream window partitions methods. *Contextual Event* stream anomalous definitions and design describes in Section 3.5 followed by described change detection procedure from the *event* stream over the each window partition in Section 3.6.

3.1. General Notation of Stream Definitions and Model

This section describes a global understanding of data stream basic notation, distributed stream processing data structure model, describes the *event* stream definitions, and notations of created *event* from IoT stream data sensors along

with designed *event* stream model.

In dynamic applications, data stream structure is represented as unbounded sequence of stream tuples and these tuples, which are mainly consisting of raw attributes records when each *tuple* is represented in a form of ⟨*x, t*⟩ pairs with formalised *implicit* or *explicit* timestamp *t*.

Definition 1 (***Tuple***): A list of data attribute/value pairs in particular data schema of $\langle s_i, \rangle$ and *t* is a discrete of tuple time stamped.

Definition 2 (**Time**): Before processing any *event* streams, *event* stream data can be structured in a time-series ordered format as $t \in T$ where *t* time is a discrete timestamp of arrived stream *tuple*. Particularly, the *event* streams can be constructed from aggregating sensor streams within three time series intervals.

Definition 3 (Data Stream): Data stream *S* is a sequence of timed *tuples* of $S = \langle (s_1, t_1), \dots, (s_k, t_k) \rangle$. Each *tuple* is ordered by timestamp *t* and can be denoted as $(t_1, t_2, \dots, t_n)$. Data stream *tuple* usually arrives at a very high rate, while in most conditions it is difficult to process or store a complete size of the data streams. Thus, alternative solution is to constructing window partitions and capture *event* streams in each window slides as describes in (Section 3.3.1)

3.2. Event Stream Model

In some of the real-world applications, *event* is resented as a single symbol without a data attribute, name or type such as , "S_1" *and* "S_2" , for two $S_1 \rightarrow S_2$ signal sensors. In *data driven* paradigm, *event* is required to comprise data type and value to construct an *event* from. Thus, in this thesis, former approach is considered to design the novel model.

Definition 4 (Event Tuple): An *event e* can be constructed from timed *tuple* (e_i, t_i) while t_i is associated with *event* time and each tuple is time stamped as $\langle e, t \rangle \in T$ .In this context, *event* model is defined as a finite sequence of $\langle s, t, d \rangle$ tuples. This can be represented as name/type (*s*), timestamp (*t*), key- value (*d*) as shown in figure 3.1. For example, consider road traffic data attribute as high-speed value (*event*) and vehicle flows per *event* tuple ⟨A, 8:10, {120,4}⟩.

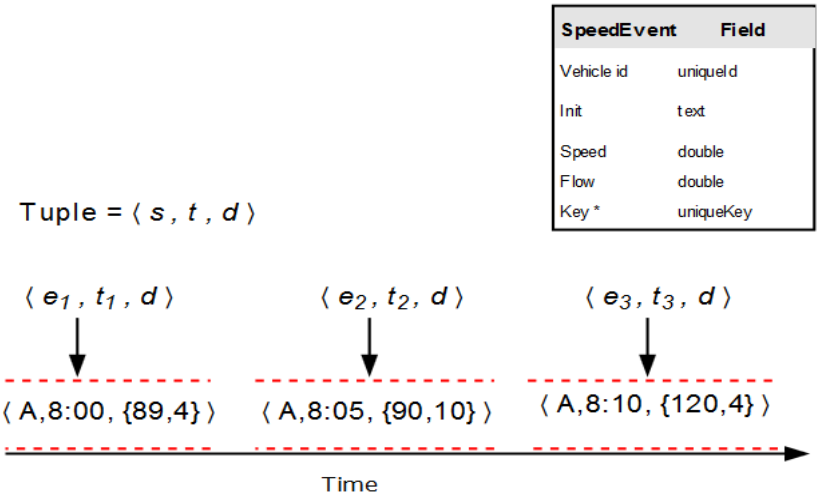


Figure 3.1: Unbounded sequence of event stream tuples.

Definition 5 (Event Streams): *Event* stream can be constructed any *S* sequence of *event* streams where each of the *event* can be represented as sequence of *event* instances or activities as denoted in Equation 3.1 and shown in figure 3.2.

$$\boldsymbol{S = \langle e_1, e_2, \ldots, e_n \rangle} \tag{3.1}$$

In dynamic stream processing model, it is practical to identify *events* automatically based on *event's* Common Correlated Attribute (CCA) value pair per each *event* stream tuples. Importantly, *event* can be considered as

anomalous behaviour within a specific context.

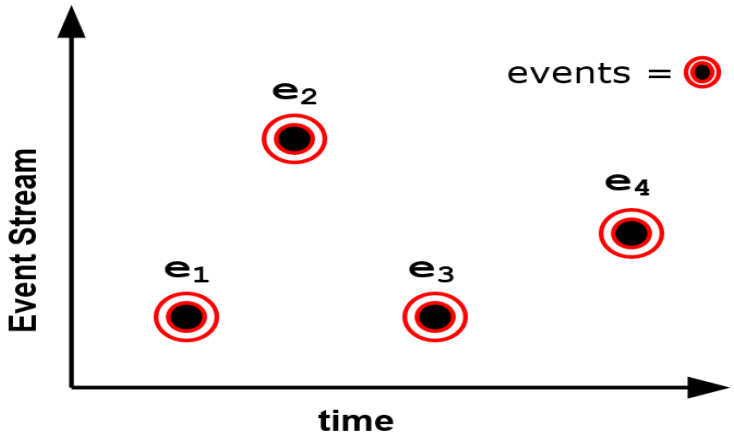


Figure 3.2: Time events interval.

Definition 6 *(*Event Time Order): time ordering in stream processing plays a significant role to differentiate between *implicit* and *explicit* timestamps of the *events* as described in Section 2.1.2. In many real-world applications, several *events* can occur together; thus, the composition ∪ of two *events* can be constructed from the *time-based* sequence *tuples* in *event* stream processing as denoted in Equation 3.2.

$$S_i(e_1, e_2) \rightarrow (e_1, t_1) \wedge (e_2, t_2) \wedge t_1 \leq t_2 \wedge e_1, e_2 \in w \qquad (3.2)$$

Overall, in both IoT application scenarios, *event* stream represents as a list of finite sequences of *events* with timestamp where *e* defines any actions with values and timestamps as defined in previous sections. The main benefit of *event* time order is to identify the time of *event, which has occurred,* and provide to semantically computational results. However, this can protect *events* from been dismissed or disregarded during the processing time and mining phases of distributed anomalous *event* detection.

Example 4: Consider *events* from $\boldsymbol{S}_1$ based on the definition 4, where each *event* is constructed from *tuple* schema of ⟨ *s, t, d* ⟩ format. The first record refers to *event* number in the S_1 (e.g., e_1), and timestamp of the arrived *event* with the *d* value, which is associated with the vehicle speed and the number of vehicle flows. The window partition can be used to collect the *events* from the sensor streams within the specified time interval T (See Section 3.2 for window partitioning concept). Figure 3.3 illustrates three *events* that have occurred in $\boldsymbol{S}_1$, where each *event* record consists of *event* number, time and speed records per vehicle. Suppose e_1 is an example of *event*, which occurred at 7:00 am with three vehicles' exceeded speed values of 125km/h. The *event* partitioning is mainly based on the CCA followed by the temporal order with tumble partition as time progress.

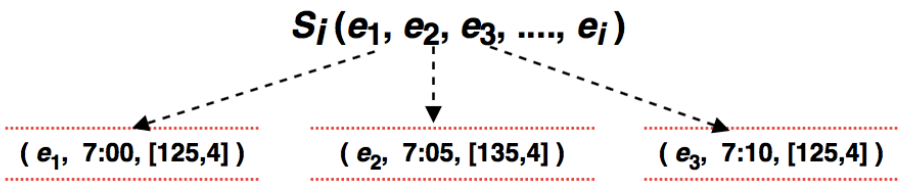


Figure 3.3: Event elements schema for a sequence of event tuples.

3.3. High Volumes of Event Stream Processing

In modern applications such as network monitoring, weather broadcast, and stock exchange, infinite streams continuously arrive at a very high rate. Thus, it is impractical to extract *events* from infinite data streams due to the constraints as described in Section 2.1.2 without prior knowledge of data format. Anomalous *event* detection plays an important role in the real time prediction. For example, road traffic officer is mainly interested in detects vehicle's over speed based on either higher (120km/h), or lower (60km/h) at a specific time period (e.g. peak, off-peak) to predict the highway traffic congestion *events*.

Similarly, officer controller monitors room temperature degree and interested in to detects unusual *event* or activates, assume room temperature is raised up to 26℃, which possibly is indicating an *event* due to either faulty sensor or the room is under on fire. However, processing, handling, and predicting anomalous *events* from large volumes of generated IoT sensor streams with a high rate can be addressed in two proposed solutions as described in the next sections.

i. **Reduction Method**: Approximation algorithm is one of the most common techniques in the data stream mining and machine learning area to fulfill the data stream constraints as described in Chapter 2. Many data stream mining methods including classification, querying, and clustering, is using a synopsis data construction and data reductionto to offer approximate answers. This is implemented by selecting a subset of data streams through micro-clusters (Charu C. Aggarwal, 2003), random sampling, (, 2007), sketches (Hao Huang, 2015), and histograms (Brian Babcock, 2002). Such solutions and techniques have been described in sections 2.3 and 2.4. The disadvantage of data reduction technique is that when data continuously arrive at a very high rate, intelligent actionable decisions are required before the *event* stream is discarded or neglected during the reduction technique. Therefore, such technique can be appropriate for the dataset in static method rather than for the data streams (Pham, Venkatesh et al., 2012).

ii. **Window Method**: a window method is a mechanism to extract relation from infinite streams and divides data into finite slices to prevent overflow of memory and concept drift (Kuncheva, 2008). For any window technique, the size and number of windows are based on two different measurements of *time-based* and *count-based*. Data sensor streams can be partitioned into according to either their arriving time, for example, partitions stream *tuples* within a specific time period **(**e.g., one-hour), or based on the number of stream *tuples* per window partitioning size of *w* (e.g. *w* =10,000 *tuples*) as

described in below. Importantly, this this method has been used in much data stream processing and mining, however, the main benefit of window partition is to control and handle change and data stream distribution. In this context, the correct implementation of window methods to detect anomalous *event* streams is a challenging task. The next sections are describing window partitioning design with detail descriptions of each window methods adaptation and justifies the most appropriate window partition method to adapt.

3.4. Event Stream Processing Window Partitioning Definitions and Designs

This section describes stream processing and window partitioning design to handle the high rate of *event* streams and manage memory overflow of the proposed computing resources.

Window Concept: Consider window *W* as constructed window partition from incoming sensor streams in length of *L* and window size of δ where *L* can be representing the length of streams based on either *time-based* or *tuples* as *count-based*. The interchange of windows is mainly depending on the sliding factor δ based on specified interval as depicted in figure 3.4. For example, 30 *event* streams can be partitioned into 3 sliding factors δ.

Time-based: Give timestamp $t \in$ T as a temporal order of the *event* stream *tuples* within specified time interval (e.g., minute, hours, days), where a *time-based* window w_t can be defined as partitioned window for arriving *event* streams according to temporal order period as described in Equation 3.3. This can be an ideal solution to partition *event* streams into time ordered *events*, for example, capture anomalous *events* per every two-hours interval window partition from the sequence of *events* list.

$$w_t = \langle e_1 , e_2 , e_3 , e_{j-1} , e_j \rangle \quad (3.3)$$

Count-based: Give $n \in N$, where n refers to the number of arrived *event* stream *tuples* (e.g., 10,000 *tuples)* from the *counted-based* technique. The notation of *count-based* can be described as in w_n in Equation 3.4.

$$w_n = \langle e_1 , e_2 , e_3 , e_{n-1} , e_n \rangle \quad (3.4)$$

In this context, the proposed window partitions can be managed and count the number of *event* streams per window slides; for example, consider counting number of high or low temperature degrees in each window partition. This is significantly very important in many dynamic application domains to detect the number of *events* per sliding windows. Overall, in this thesis, window partitioning method is consideration is an appropriate solution to be adopted in both traffic monitoring and temperature scenarios to handle high volume of *event* streams from IoT sensors and to prevent change or concept drift within the data stream distribution. As a result, it is more practical to compute *event* stream in real time before such *events* disappear; consider an example of traffic stream sensors that capture vehicles' over speeds according to the speed limitation or congestion speed in certain location. Importantly, the result of *event* steam can be grouped into one of the aforementioned three windowing methods. Window method is also implemented to handle data streams in many streaming application domains, such as stock exchange or weather broadcast (Tanbeer, Ahmed et al., 2009). The main advantage of window method is to handle high volumes of the data stream in terms of scalabilty by partitioning data streams into windows of slides based on sliding ***window, landmark window,*** or ***tumbling window***. A detailed description of each window method and proposed examples are demonstrated in the next Sections 4.4.1., 4.4.2., and 4.4.3. respectively.

3.4.1. Sliding Window Definition

Sliding window is one of the most common methods and mechanisms to handle incoming continuous *events* streams (Li and Lee, 2009). The data structured concept of sliding window is primarily based on First-In-First-Out (FIFO) technique. In the last decade, wide number of studies investigated on the way of how to handle, learn, and monitor the data streams. These studies mainly proposed sliding window method including FLOAR, and ADWIN (Bifet et al., 2006). For example, in ADWIN algorithm the proposed sliding window to keep contains the stream length from the most recent data streams partition and discards the old data streams since the algorithm scans the learning tasks.

The main advantage of sliding window is the guarantee of the data stream in the memory space in the window size (Bifet, 2009). Additionally, sliding window facilitates to monitor the data distribution and changes within the data stream (Bifet 2009, and Brzezinski & Stefanowski, 2014). In order to handle the high rate of streams, sliding window method is widely proposed in many real life applications including stock exchange (Babcock, Datar et al., 2002), fraud detection (Kuncheva, 2008), medical diagnose (Amineh Amini, 2014), intrusion detection (Vu et al., 2014), network sensor nodes (Hoens, Polikar et al., 2012), weather streams (Dariusz Brzezinski, 2014), and social media streams (Hoens, Polikar et al., 2012). The main disadvantage of the sliding window in IoT traffic anomalous *event* detection is the replication of the *events* in each window this concept is more discussed in Example 1.

Sliding Window: consider sliding window of either w_t or w_n, where t refers to the time interval of arrived streams and n stream *tuples* per window. The window updates with bounded size when new *event* streams arrive until $L = \delta$ is satisfied as described in Example 1 and figure 3.4.

Example 1: Suppose, n number of anomalous *event* streams for the last ten minutes when t = 10 seconds as shown in figure 3.4. In this scenario, window

w_1 consists of e_1 to e_5 from $t_0 - t_5$, while w_2 consists of e_4 to e_8 from $t_4 - t_8$ since, e_4 and e_5 *events* are belonging to both w_1 and w_2. In this context, replication between two windows has occurred and this can have significant impact on the computational results. Thus, this is a major drawback in real time *event* stream detection when it is impractical to have duplicated *events* within new constructed window; hence, when w_2 is completed *event* e_1, e_2, and e_3 will be disregarded.

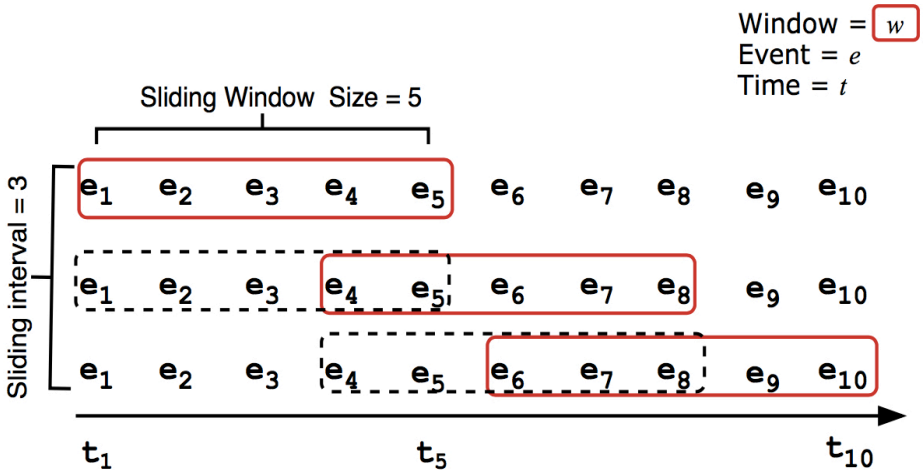


Figure 3.4: An example of event streams partition in sliding window.

3.4.2. Landmark Window Definition

Landmark window is known as fixed upper and lower bound window approach; this scenario constructed window includes the complete n number of *events*. Window starts from a particular point and expires when the size of the window is completed. The window size monotonically increases as time progresses and in this situation; it is impossible to discard any *events* due to the predefined length of window size. For such reason, landmark window is inappropriate to adopt for anomalous *event* stream processing due to the increasing number of streams within each window; thus, this leads to memory

overflown and highlighted as major drawback of the landmark window. On the other hand, *event* replication can certainly occurs when the size of *events* increases (Tanbeer, Ahmed et al., 2009).

Landmark window: Suppose landmark window W is constructed from $W = \{w_1, w_2, w_3, \ldots, w_n\}$, where w_1 is as first constructed window which consists of *event* streams while the current length of W progressively changes with new incoming *event* streams within the landmark in w_2, and w_3 respectively.

Example 2: Assume sequence of *event* streams can be added into number of windows as shown in figure 3.5. Where the first w_1 starts with eight *events*, and the state of the current window is changed since new *event* e_9 is added into w_2 and w_3 progressively; hence, the size of window expands as time progresses, particularly, when new *event* streams continuously arrive from the sensors. For example, w_1 starts from $\boldsymbol{t_1} - \boldsymbol{t_8}$ and holds *events* from e_1 to e_8; similarly, window w_2 starts from the same point of $w_1$with adding extra e_9,and similar procedure is repeated for w_3 respectively. However, landmark window is one of the classical window models. The main drawback of this method is incapability of handling high volumes of *event* streams, due to expanding window size as new *event* streams continuously arrive. Thus, this method is leading to allocated and consumes more memory space and is time-consuming for the computational results.

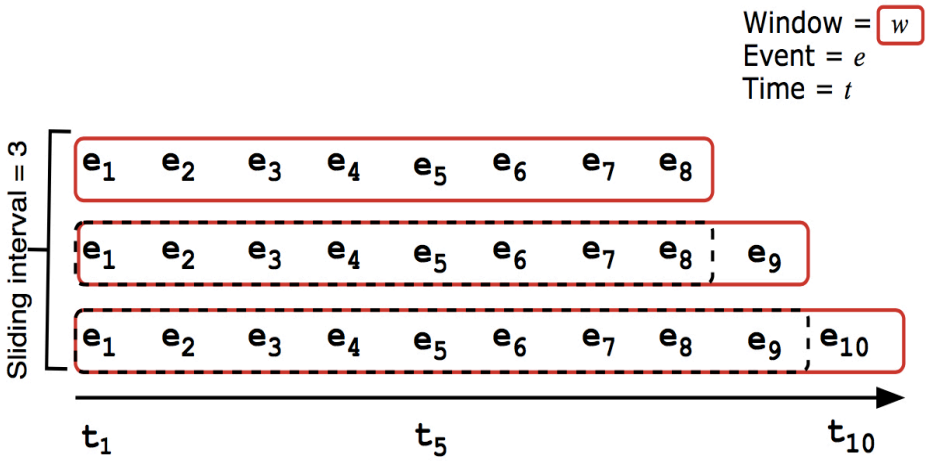


Figure 3.5: An example event stream partition in landmark window.

3.4.3. Tumbling Window Definition

Tumbling window is mainly relying on the size and segments of the *event* streams, and the primary aim of this method is to define a specific time interval before each window becomes full of its capacity. This can be computed by number of *event* streams that arrive within one hour time interval and start new tumbled window straight after the previous window tumble is becoming full (Manish Gupta, 2014). It is more practical to use a small size of the window in order to achieve accurate computational results as using larger size of tumbled window is more difficult for the computational results due to the time constraint per each window. An ideal solution is to monitor *event* stream states within window partitions to control and handle each size of window partitions. Importantly, the main advantage of tumbling window is the impossibilities of *event* replications; for example, a specific *event* (e_1) can only exist in one window (e.g., w_1) only. The disadvantage of tumble window is their dependability on the size and sliding bounds of the predefined window; however, such constraint can be addressed by monitoring the state of changes per window partitions as described in Section 3.6.

Tumbling Window: Let w_t be a tumbling window size of *event* streams according to the satisfied conditions of $L = \delta$ window length. The window size possibly can shift based on the predefined time, for example, after one hour is terminated, a new tumbled window can be constructed. In this context, *event* streams can be partitioned over n number of tumbling windows as described in the next section.

Example 3: Let compute over speed stream tuples in road traffic scenario, where *event tuples* can be grouped or paired according to each vehicle's speed values (tuples) within each tumbling partition based on either *count –based* or *time-based* techniques. Figure 3.5 demonstrates the computing process for the number of vehicles (over speed tuples) within each $\boldsymbol{S_1}$, $\boldsymbol{S_2}$, $\boldsymbol{S_3}$ every ten minutes. Consider, window w_1 consists of *events* from e_1 *to* e_5 from $\boldsymbol{S_3}$ and window w_1 expired at $t = 5$ when w_2 is constructed for new *events* partitions from e_6 to e_{10}.

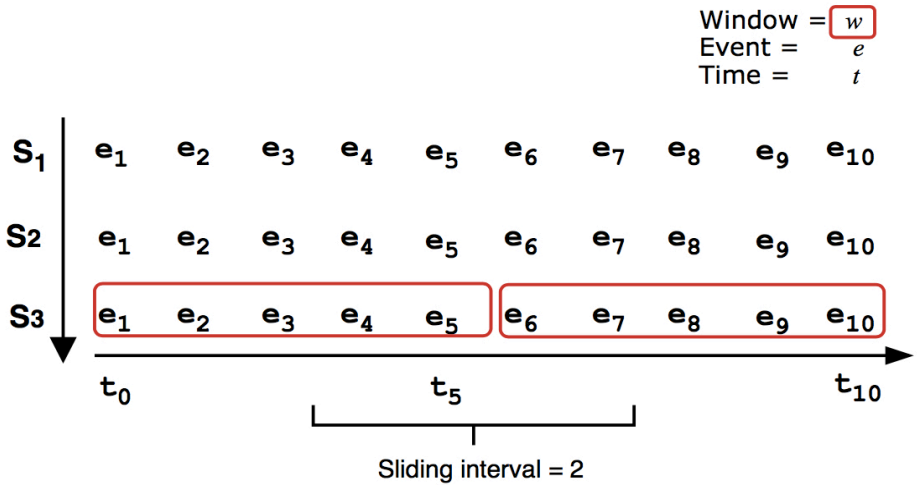


Figure 3.6: An example of event stream partition in tumbling window.

Overall, since IoT streams are continually arrives at a very high rate, tumbling window partition can be an appropriate solution to be adopted in order to handle such high rate of streams and control the changes within *event* stream data distributions. A detailed description of the designed method of window partition is discussed in the following sections.

3.5. Event Stream Window Partitioning Design

As described in the previous sections, the main key challenging task is to select appropriate method to process high volumes of sensor streams and detect anomalous *events* dynamically. This can be achieved by designing and implementing new *event* streams window partitions in parallel based on distributed data stream structured model. In this context, *event* streams within the predefined window can be computed in parallel across the number of computer nodes according to their correlated stream tuple values. Thus, one of the most appropriate window methods to propose and design *event* stream partition is a tumbling window. This can be achieved by partitioning *event* streams into number of equal constructing window length and computes the final results per each window partition. The *event* streams per each window partition can be grouped based on their correlated stream *tuple* values as described in Section 2.2.2 and figure 3.6 where w_i is $_i^{th}$ number of window partition which is constructed from number of *events* within the *event* stream time interval from $\mathbf{t_1}$, to $\mathbf{t_{10}}$.

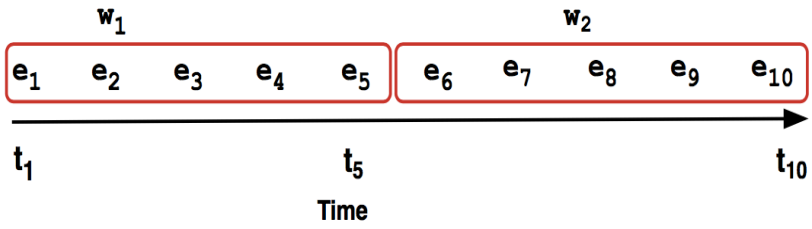


Figure 3.7: A general design sample of tumbling windows partitions.

A detailed description of window partition notations and the structural design model is descried in the following definition.

Definition 7 (Window Partitioning): The semantics of window partitioning design w_p can be based on the number of *event* stream *tuples* and divided

events into a window-based model until the window condition length is satisfied as denoted in Equation 3.5 where w_i represents the number of windows (e.g., w_1, w_2), p partition, and $w_i \subseteq S \; while \; w_i \cap w_j = \emptyset$.

$$S_{wp} = \bigcup_{i=1,k} w_i \tag{3.5}$$

The main procedure of such concept is described in algorithm 1 and 2 for each count or time-based method. The notation of S refers to the number of sensor streams, k as i^{th} length of window partitions from the *event* streams according to either *time-based* or *count-based* partition and such parameters can be defined as w_t or w_n in each algorithm.

In the *event* stream window partitioning scenario, *count-based* refers to the number of *events* per window and it can be so-called *event-based* window and denoted as w_n. Since *time-based* is associated with the time interval length of window partition directly constructed from the event streams, such approach can be called *event-time* based windows and it can have denoted as w_t. The design and procedure of window partition for *event* stream is categorised in two steps: defining window condition and computing *events* per window partitions.

Algorithm 1 describes the first step to initialise window based from the sequence of *event* stream according to a predefined window (lines 1-3). The second step is directly constructed from continuous arrived *event* streams for processing; the algorithm computes the n number of *events* per each window partitions. Then the *events* can be grouped into a new window partition, for example, constructing first window w_1 from 1,000 *event* stream *tuples* (lines 6-8). The algorithm checks for the size of window length, if the size of n *event* stream is larger than current window, the new *event* stream can be assigned into the next window, which could be w_2 (line 10-12). The design of window

partition in algorithm 1 is based on two steps; first, the *Input Event Stream Processing* which is associated with arriving sequence of *event* streams from the DSPE and second, the Output Event-based Windows, this is mainly refers to the number of tumbling window partitioning over the sequence of *event* streams from the *event* stream processing step as illustrated in figure 3.7.

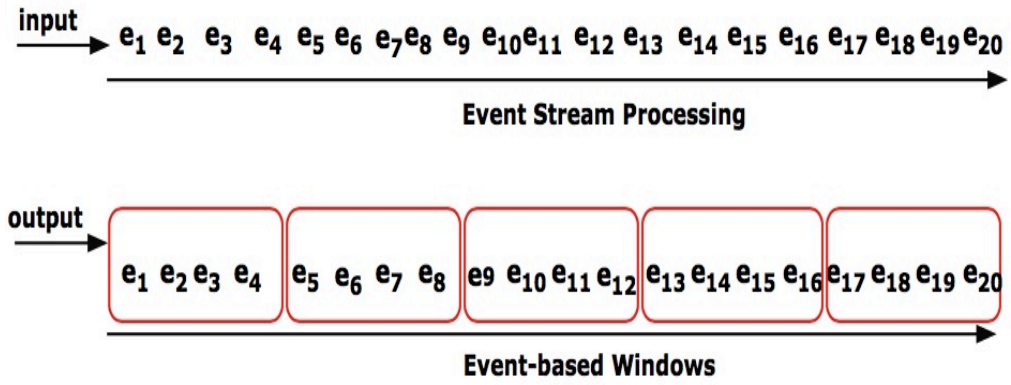


Figure 3.8: Count-based event stream window partitions.

Algorithm 1: *Count-based* Window Partition w_p

Input: *event* stream *tuple* ***S***

Output: count-based w_n

1. Initialise window *L*
2. Initialise the complete condition
3. List <*event* stream *tuples*> window ***w***= { };
4. **foreach** stream arrival *tuple* $\boldsymbol{e_i} \in \boldsymbol{S}$ **do**
5. $\boldsymbol{w} \leftarrow \boldsymbol{w} \cup \{ \boldsymbol{e_i} \}$;
6. **if** $|\boldsymbol{w}|$ = *count-based* **then**
7. w_n = window size (capture 1,000 *event* stream *tuples*)
8. window.add (*n*)
9. **else**
10. **if** $n > w_n$ length **then**
11. update $w_2 \leftarrow$emit *events* to new window
12. return w_n
13. **else**
14. **end**

The design of *time-based* window partitioning can be constructed from algorithm 2. In this context, algorithm 2 is first initialising the window according to the arrival of the *event* streams at *t* where constructed new window time-based partition can be denoted as w_t (lines 1-3). The algorithm checks for every *event* stream based on their ordered timestamp (lines 4-6); if the *events* from within the window partition are is *time-based*, then such *event* streams can be grouped into a number of *time-based* according to the time ordered sequence of $\mathbf{t_1}$, $\mathbf{t_2}$,…, t_n (lines 6-8). For example, *event* streams from $\mathbf{t_1}$, $\mathbf{t_2}$, can be emitted to w_1, w_2, respectively.

Algorithm 2: *Time-based* Window Partition w_t

Input: *event* stream *tuple* $\boldsymbol{S}$

Output: time-based w_t

1. Initialise window *L*
2. Initialise the complete condition
3. List <*event* stream *tuples*> window $\boldsymbol{w}$= { };
4. **foreach** stream arrival *tuple* $\boldsymbol{e_i} \in \boldsymbol{S}$ **do**
5. $\boldsymbol{w} \leftarrow \boldsymbol{w} \cup \{ \boldsymbol{e_i} \}$;
6. **if** $|\boldsymbol{w}|$ = *time-based* **then**
7. $\boldsymbol{w_1} = \boldsymbol{t_1}$ (*event* streams from e.g., 7:00-9:00)
8. w_t = timestamp $-\boldsymbol{t_1}$

   else
9. **end**

As showing in figure 3.9, the *time-based event* stream window partitions procedure is organised in two steps; First, the input *event stream processing* step is associated with the number of *event* streams that have arrived from $\boldsymbol{S_1}$, $\boldsymbol{S_2}$, and $\boldsymbol{S_3}$ at between 7:00am to 24:00am hours. Second, the output of *event-time window* step is associated with the number of window partitions according to their constructed timestamps. In the later step, *events* have been partitioned based on the predefined *time-based*; for example, window can be constructed from the arrived *events* at between 7:00am to 9:00am at $\mathbf{t_1}$ for the first window partitioning w_1; similarly, new window w_2 can be automatically constructed from *event* streams at between 11:00am to 13:00am within t_2 based on the *time-based* sequence ordered as $\boldsymbol{t_1}$, $\boldsymbol{t_2}$, $\boldsymbol{t_3}$, $\boldsymbol{t_4}$, and $\boldsymbol{t_5}$ respectively. *Event* streams can be classified and grouped based on the *FieldGrouping* mechanism in the DESP. This technique is mainly depending on the correlated stream tuple value and the size of time-based partitioned window; for example, $\boldsymbol{S_1}$ consists of e_1 to e_4

according to their time interval processing where each time t_1, t_2 is respectively constructed. Importantly, the main challenging task and benefit of time-based window partitioning is to process data streams as fast as achievable before *event* streams are discarded. In this situation the learning computational output result can be more accurate and achievable.

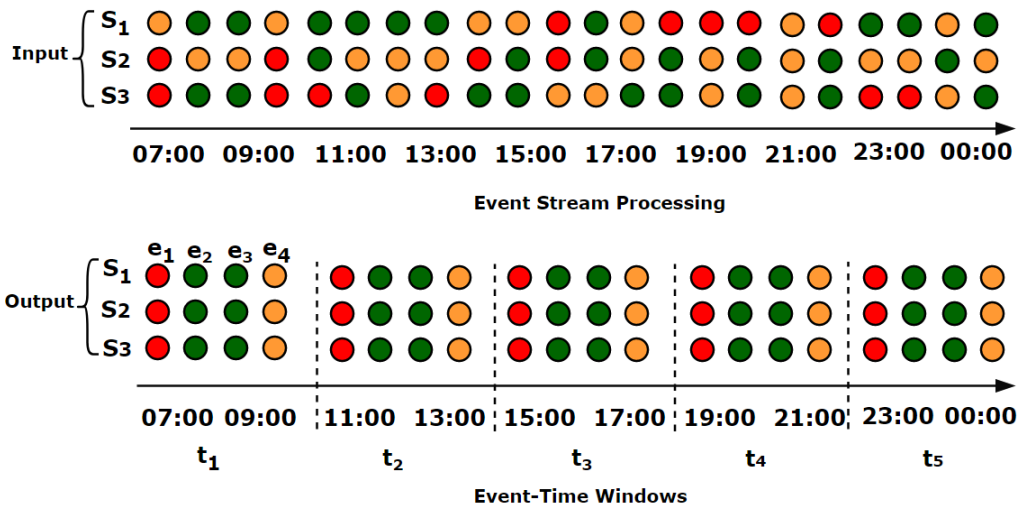


Figure 3.9: Time-based event stream window partitions.

3.6. Contextual Event Stream Definitions and Design

A normal technique of *Contextual* anomaly is to define the anomalous attribute within a specific context as described in Section 2.1.2. Such behaviour is referred to sequential analysis in other application domains when data is in a static mode. Several studies including Yexi Jiang (2014) and Saleh, Hagedorn et al. (2015) stated that *Contextual* anomaly is the most appropriate method to detect *Contextual* behaviour of the data streams. Jiang (2014) argues that in dynamic situations *Contextual* method is possibly produces less accurate computational result from the data stream due to stream constraint and characteristics. Thus, to deal with such problem, in this thesis, the window

partitioning is regarded as a solution to address a speed rate of the data stream and to discover changes in the stream.

Definition 8 (Contextual Event Stream): *Event* stream is anomalous in a precise context, when such behaviour is not normal in a different context. According to Angiulli, Fassetti (2010) and Chandola, Banerjee et al. (2012) *Contextual* anomaly refers to the change in the context attributes position within the *event* sequence. Consider sequence of *event* streams from $\boldsymbol{S}_1$, $\boldsymbol{S}_2$, and $\boldsymbol{S}_3$ are arriving from road traffic sensors respectively; speed *events* less than < 60 km/h at 7:00 am are considered as normal behaviour due to the traffics movement. On the contrary, similar speed at midnight indicates an accident due to the traffic speed is slowing down and can be considered a *Contextual* anomalous *event*. Importantly, this concept is based on the change in the *event* behaviour while the context of *event* attribute is remaining the same. In this thesis, such concept is beneficial to design C_A model and implements in Contextual Event Stream Anomaly (CESA) algorithm over the *event* streams in real time. Consider road traffic scenario when sequence of *event* streams which have been emitted into DSPE and grouped into sequence of stream *tuples* according to their timestamps $\boldsymbol{t_1}$, $\boldsymbol{t_2}$ as illustrated in figure 3.9 where the first *tuple* consists of *event* number, *event* occurrence time and values. For example, e_2 in $\boldsymbol{S}_2$ consists of 10 over speed records of 140 km/h at $\boldsymbol{t_1}$ (8:45 am) and similar behaviour e_2 in $\boldsymbol{S_2}$ at $\boldsymbol{t_2}$ (22:10 pm) consists of over speed values; hence, such over speed at midnight can be considered as *Contextual* events which is due to an accident on the road or suspicious activity.

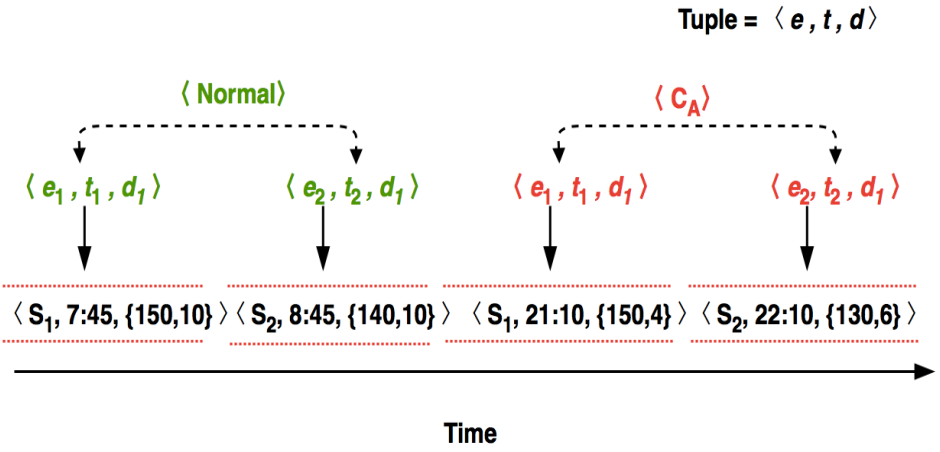


Figure 3.10: Sequence of events with normal and contextual behaviours.

The *Contextual* Anomaly $\boldsymbol{C_A}$ model can be denoted as a sequence of *event* stream *tuple* partitions as described in the Equation (3.6) in the Contextual Event Stream Anomaly (CESA) algorithm 3 where S_i is the i^{th} *event* stream from the collection of sensor streams during window tumble partitions within (w_{t+1}, w_{t+2}, ...,) and V_i is associated with the *event* stream scores of $\boldsymbol{S}_i$ from *AScore* as described in the next section.

$$C_A = \{S_i, [w_{t+1}, w_{t+2}], \dots, V_i\} \tag{3.6}$$

Definition 9 (Contextual Event Score): The output of anomalous *event* is associated with the result of *event* streams computed by using a scoring technique for every *event* in $\boldsymbol{S}$ according to the C_A in the sequence format. In this situation, algorithm 3 reads *event* streams and the rule set *r* checks for the CCA (See Section 3.3.2) to find any rule that covers the sample of $\boldsymbol{S}$. The probability of each given *event* stream *v* value is expected to be positive when *event* attributes e_i in S_i is less than $P\ (e_i = v|r) < 0.5$, in contrast, if the is $P\ (e_i = v|r) > 0.5$ the value of the *event* stream score is expected to be a

negative value. However, r rule computes from the sequence of *r* and for the purpose of *Contextual* behaviour. The *AScore* computation is denoted in Equation 3.7 and 3.8.

$$AScore = \frac{1}{d}\sum_{ei=1}^{d} \log\left(\frac{P\ (e_i = v|r)}{1 - P(e_i = v|r)}\right) \tag{3.7}$$

$$\frac{1}{d}\sum_{e_i=1}^{d} \log(P\ (e_i = v|r)) - \log(1 - P(e_i = v|r)) \tag{3.8}$$

A detailed description of the evaluation performance for *AScore* is presented in Section 5.6. Algorithm 3, describes the Contextual Event Stream Anomaly (CESA) process the idea of the rule set structure is used in many data stream research studies including in Duarte, Gama et al. (2016).The algorithm starts with an empty rule set r = {}.When new *event* streams are partitioned by the window in algorithm 1 and 2, CESA algorithm checks to find out if the *event* stream is covered by C_A model. For every *event* stream e_i in $\boldsymbol{S}$, each rule set is required to be checked and computed based on the Equation (3.10) for α changes in the *event* streams. If probability of any *event* stream value according to the context attribute e_i is changed, the rule set can be removed and the value of *event* stream within the C_A can be updated. On the other hand, to check the accuracy of the C_A, the algorithm will assign *AScore* [0,1] to each *event* stream *tuple* in $\boldsymbol{S}$, and the value of each sensor data $\boldsymbol{S}$ can be updated and nominates C_A based on the *event* context value in the sequence.

On the contrary, if e_i is covered by the rule sets and not considered as *Contextual* Anomaly $\boldsymbol{C_A}$, the *PH* test computes the error *e* based on the α

magnitude of changes and updates the *r*. Thus, if any *event* stream e_i is covered by RS according to the C_A the algorithm then returns *Contextual* Anomaly $\boldsymbol{C_A}$ values based on the predefined threshold. For example, in road traffic scenario, the output of the CESA is either [0,1] where 0 is associated with speed *event* during the normal hour at 10:00 am, and 1 as *Contextual* anomalous *event* at 23:00 pm. Such concept is based on the *event* stream context value (e.g., *event* with similar speed as 120km/h at 10:00 am can be considered as a *Contextual event* at midnight 23:00 pm).

Algorithm 3: Contextual Event Stream Anomaly (CESA)

Input: *Event* Streams *S*

S: $<e_1, e_2, \ldots, >$

Output: *Contextual Events Stream*

1. Init Rule Set *RS* = { }
2. $C_A \leftarrow e_i$covers by model using Eq. (4.11)
3. **foreach** *event* stream $e_i \in S$ **do**
4. **foreach** rule $\boldsymbol{r} \in RS$ **do**
5. computes *S* using Eq. (4.10)
6. **If** $\alpha \leftarrow$ detected, **then**
7. remove ***r***
8. update V_i get the value in $\boldsymbol{C_A}$
9. **end**
10. **If** no rule selected *in RS* **then**
11. update default *RS*
12. **end**
13. **if** *AScore* $(e_i) = \lambda$ score using Eq. (4.10) **then**
14. ***RS***← update *AScore*
15. **end**
16. **if** e_i not covered by *RS* and e_i is not in $\boldsymbol{C_A}$ **then**
17. calculate prediction *e*
18. Test *PH (e, λ)*
19. **end**
20. **else**
21. update e_i when *RS* match $\boldsymbol{C_A}$ **then**
22. return $\boldsymbol{C_A}$= [0,1];
23. **end if**
24. **end**
25. **end**

3.7. Event Stream Change Detection

Event stream is evolving from time to time; hence, it is impractical to assume that *event* streams have the same probability of data distributions (Gama et al., 2009). For example, if a sets of data streams in t_1 and t_2, timestamps are different, and then this indicates the occurrence of a change within the sequence of the *event* stream partitioning. One of the aims of this thesis is primarily to handle and address distribution changes over the window partitioning methods. In this context, the appropriate algorithm is required to incrementally adapt to test newly arrived streams and to be able to compute the probability of *event* stream behaviour in parallel. Thus, change detection method can be an appropriate solution to monitor the *event* stream status over each window partitioning.

In recent years, the problem of change detection has been studied intensively as it has been recognized as one of the most common problems in the streaming applications (Tran, Gaber et al., 2014). Since then, several studies have attempted to address this problem; for example, (Kuncheva, 2008) proposed supervised learning such as ensemble classifier to address change detection problems over the data streams. Alternative meta-algorithm of measuring change detection in the data streams based on non-parametric statistical distance computation is proposed in (Daniel Kifer, 2004). Similarly, Farid, Zhang et al. (2013) proposed adaptive ensemble classifier approach to predict a novel class detection concept changes from sequence of infinite of data streams.

Change within sensor network is proposed by Tran, Gaber et al. (2014), and according to this study, the most reliable method to detect and monitor change over high volumes of the data stream is distributed processing which deploys online incremental learning algorithm. On the contrary, it is impractical to handle and detect changes from high volumes of data stream centrally; this is due to the resource constraint and data structure or computational model. Yang and Fong (2015) proposed single tree learning classifier to discover the changes

from streams; however, this approach is primarily designed to address a specific problem. For example, in supervised learning, the problem of model change or update and novelty change over data stream is the most common problems (Ikonomovska et al., 2010; Bondu & Boulle, 2011; Badarna & Wolff, 2014). Similarly, in unsupervised learning, change detection has been studied to detect data stream cluster's behaviour (Vallim & De mello, 2014; Demšar & Bosnić, 2018).

In general, change in data stream occurs in two situations: first, during the prediction model if the model is stable; otherwise data stream is deviated during reconstructing of the model. Second, change possibly occurs during window's either being disjoined or its reconstruction when new *event* data stream arrives, and the *event* stream value can be dissimilar in comparison to the pervious window *events*. Assume the value of *event* e_1 in $(\boldsymbol{w_1}, \boldsymbol{t_1})$ is (200km/h) which is different from the similar *event* in $(\boldsymbol{w_2}, \boldsymbol{t_2})$ due to the change in the traffic speed behaviour. In this situation, both conditions can be concerned due to the uncertainty of the *event* streams from each window partitioning and model prediction. The main drawback of change within window models are studied in Tao and Ozsu (2009); while in dynamic situation, detecting change from the *event* streams are challenging tasks. Thus, appropriate solution is to design window change detection method based on *partition window* w_p with time interval Δ_p, and *monitor window* w_{m} with time interval Δ_t. This can be achieved by using w_{m} which represents as a tumbling window and the result of window partitioning can be presented in *merged partition windows*.

3.7.1. **Window Change Detection Definition**

In this section, a novel change detection design will be proposed based on the number of unexpected *event tuples* in each *partition window* w_p and such concept is defined as follows.

Definition 10 (Window Change Detection): Let $d: w_1 and\ w_2 \rightarrow \mathbb{R}$ be two *partitioning windows* for two *event* streams from $S_1\ and\ S_2$ in *S,* where *d* associates with the dissimilarity between both partitioning $w_1 and\ w_2$.

In this context, changes can be quantified based on the dissimilarity of two *window partitionin*g. For example, consider dissimilarity distance *d* as " $if\ d\ (w_1\ , w_2\ ) >\ \varepsilon$ ". The dissimilarity between two *window partitioning* can be represented in $w_1(e_1, e_2, e_3, .., e_n)$ and $w_2(e_1, e_2, e_3, .., e_n)$ and change can be expected when two *window partitions* are contains dissimilar *event* stream tuples. Hence, this can be measured by the most common Euclidian distance metric as denoted in Equation 3.9 where *p* is the associate with *window partitioning* w_p, and algorithm 4 describes the process of *event* change detection.

$$d(w_1, w_2) = \sqrt{\sum_{k=1}^{p} (w_1, w_2)^2} \tag{3.9}$$

Assume S_i consists of a sequence of *event* streams $< e_1, e_2, e_3, ..., e_n >$. The algorithm is first defining the *event* stream condition and begins with constructed window from *event* streams, which are arriving according to their timestamp t , and the size of i^{th} *window partitioning*. As new *event* emerges in w_2 and for each update the algorithm checks for the dissimilarity to see if $d\ (w_1, w_2) > \alpha_i$. The selection of α_i is depending on the distance length of each *window partitions*, if length is $d > \alpha_i$ larger, then the probability of false alarm is high and new window can be constructed. In contrast, smaller change can be detected in the distribution if several *event* stream behaviour is also dissimilar. The detail description of window changes detection performance results and evaluation metrics are described in Chapter 6. In data stream processing and mining, a key strategy of window change detection is to match two *window partitions* (w_1, w_2) and measure the change based on their distribution rates. Consider a problem of change as null hypothesis of H_0 compared with other hypothesis of H_1 for two

window partitions dissimilarity as shown in Equation 3.10 where $d\ (w_1\ , w_2\ )$ is the distance function which computes the dissimilarity of the (w_1, w_2) window partitions

$$\begin{cases} H_0 & d\ (w1, w2)\ \leq w \\ H_1 & d\ (w1, w2) > w \end{cases} \tag{3.10}$$

In this situation, the assumption of change in each window partitions are considered where each hypothesis can reflect on the dissimilarity of the window partition based on the size of window partitions and timestamps of the *event* streams.

Algorithm 4: Detect Change (*S, D*)

Input: ***S*** *event* stream

d*:* dissimilarity *d**

1. **Stage 1***:*
2. **begin**
3. $\boldsymbol{for}\ t1 = \ current\ window\ \ \boldsymbol{do}$
4. $w_1, i \leftarrow$ first w_p
5. $w_2, i \leftarrow$ second w_p
6. **end**
7. **Stage 2:**
8. **while** $not\ at\ the\ end\ of\ stream$ **do**
9. **window** $w_{2,i}$ by 1 *event*
10. **If** $d\,(w_{1,i,}\, w_{2,i}) > \alpha_i$ **then**
11. $t \leftarrow$ current time
12. Declare change at $\boldsymbol{t_1}$
13. Clear the windows and GOTO stage 1
14. **else**
15. Move $w_{2,i}$ to hold new *events*
16. **end if**

3.7.2. Change Detection Standard Evaluation Measurement

The most common method to measure evolving the data streaming is described in below.

First, consider two *event* streams are created from two sensor streams *S*, where each *S* can be a stream with sequence of *event* stream tuples. Change possibly occurs when the timestamps of two streams (S_{t+1}, S_{t+2}) are different according to $\boldsymbol{t_1}$and $\boldsymbol{t_2}$ as denoted in Equation (3.11). This is due to dissimilarity of incoming *event* stream value at $\boldsymbol{t_1}$ compared to the *event* stream values at t_2.

$$\sum^{S1}(t_1) \neq \sum^{S2}(t_2) \tag{3.11}$$

Second, consider a sequence of *event* streams $e_1, e_2, \ldots, e_n$, in each *window partition* constructed from timestamp t . To measure the rate of change over such *event* streams, *Page-Hinckley* (*PH*) test is considered to be an appropriate method; since the first assumption is depending on dissimilarly of sensor behaviour and the accuracy of learning detecting is very low (Daniel Kifer 2004). Importantly, *PH* is relying on accumulated sum of a loss function error for the sudden change based on using Gaussian signal. This validation is realistic to propose and to measure *event* stream change detection. In this thesis, *PH* is considered as an appropriate test validation method to monitor online changes from the *event* stream sequence. Changes in *event* stream are common and detection validation can be assumed according to the prediction error rate as rule of $\widehat{e_{t+1}}$. *PH* is capable to monitor change from current *event* streams based on predicted error rate and pervious *event* streams. Thus, the abnormalities between such *event* streams *tuples* are can be considered as an error. The computational error rate also is computed based on two loss functional computation errors metrics of Mean Absolute Error (MAE) and Root Mean Square Error (RMSE). The predicting error rate can be defined when the model does not cover the rule set. Similar techniques have been used in (H. Mouss

2004). This concept is based on testing sum of cumulative m_T from the sum of *events* $<e_1, e_2, \ldots,>$, the sum of differences between the observed e_i and their mean can be set within time interval [1, t] and change is expected according to Equations 3.12 and 3.13 where m_T is associated with the maintaining of the minimum test m_T $(m_t, t = 1,. . ., T)$.

$$\overline{e_t} = \frac{1}{t}\sum_{t=1}^{t} e_t \tag{3.12}$$

$$m_T - \frac{1}{T}\sum_{t}^{1}\{e_t - \overline{e_t} - \alpha\} \tag{3.13}$$

T is associated with monitoring test for the number of *event* streams based on t ∈ T where α also refers to the change degree for every e_i in *t*. The threshold parameter λ can be set to observes $m_T \in [0, 1]$, in $(e_1, e_2, . . ., e_n)$, and α can be referred to the magnitude of changes when *PH test* computes the difference between $PH_T = m_T - M_T$. However, the λ depends on the main false change rate prediction error. Hence, increased or decreased the false rate is mainly depending on the predefined threshold over the *event* streams to test the change detection.

Chapter Summary

This chapter has described the research problem and definitions of the *event* stream based on the distributed stream processing data structure model and the designed process of detecting high volumes of anomalous *event* detection from unbounded sequence of sensor streams. Theoretically, processing and detecting a high volume of event stream in real time is requires an appropriate method in order to handle change within the *event* streams. In this situation, we defined appropriate method to capture and partition unbounded sequence of *event* streams into a number of window partitioning before the *event* stream is evolved. Since we argued that data stream is dynamically changing, we designed tumbling window partition as a most reliable and alternative solution to handle the state of the *event* streams. The *event* streams from windows are further processed to detect changes and recover from missing *event* stream tuple. The *event* streams then are aggregated to detect *Contextual* behaviour form the streams based on the anomalous *event* score rates. A detailed implementation of described assumptions and method are demonstrated in the next chapter.

Chapter 4

Distributed Contextual Anomaly Detection (DCAD)

Contextual anomaly detection from sequence of unbounded *event* streams has proven to be a new research challenging tasks to predict unexpected behaviour of *events* in the different context. The complexity of the *Contextual event* reflects on the robustness of the proposed method and algorithm. In this chapter, we propose a novel and effective solution of distributed *Contextual* anomaly detection method to identifies different *Contextual* behaviour of *events* over the sequence of sensor streams. Section 4.1. demonstrates the overview of DCAD method. Section 4.2. will address the implementation of *event* stream collection module in the DCAD. The implementation of the DCAD approach including parallelism concept is explained in Section 4.3. The proposed DCAD architecture is described in 5.4. Section 5.5 is devoted to distributed *event* stream detection in parallel.

4.1. An Overview of the DCAD

The proposed distributed *Contextual* anomalous detection is based on the computational constraints and limitation as described in pervious Chapter 3. Importantly, the main difference between existing anomaly detection and

DCAD is that DCAD can be constructed from the number of stream modules according to the divide-and-conquer approach based on three decomposed of distributed modules; a) Event Stream Collection; b) Event Stream Processing; and c) Contextual Anomaly Detection. The existing anomaly detection methods are mainly considering centralised detection for data aggregation, which is beyond the scope of high volumes of sensor *event* data streams detection. This is due to the need of continuous update of the *event* stream model online and unexpected change to the *event* streams. A detailed description of each module is described as follows.

4.2. Distributed Contextual Event Stream Processing and Detection

This section is describing the high throughput of *Contextual event* streams processing, partitioning, and detection designed method. In this section, *event* streams are representing as unbounded sequence of *tuples*. For example, figure 4.1 demonstrates the formats of unbounded sequence of *events* from two IoT applications (*Temperature* and *Speed*), which have been, emitted form distributed messaging system for the processing, partitioning, and detection.

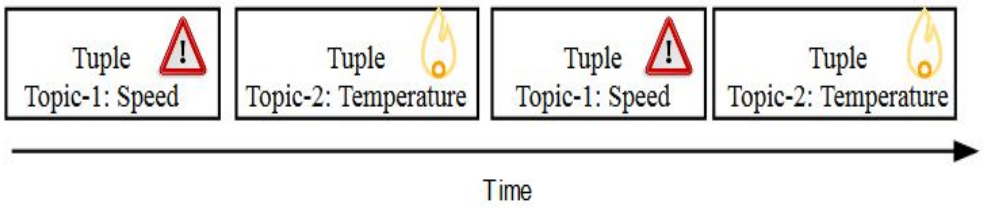


Figure 4.1: Input event stream tuples.

After *event* streams have been aggregated from distributed messaging system, then the distributed *Contextual* anomalous *event* stream topology is constructed based on the Directed Acyclic graph (DAG) data structure model. In this

context, DCAD is mainly decomposed of three main modules of pre-processing, matching *events* and *Contextual* detection as depicts in figure 4.2 and Appendix 2.

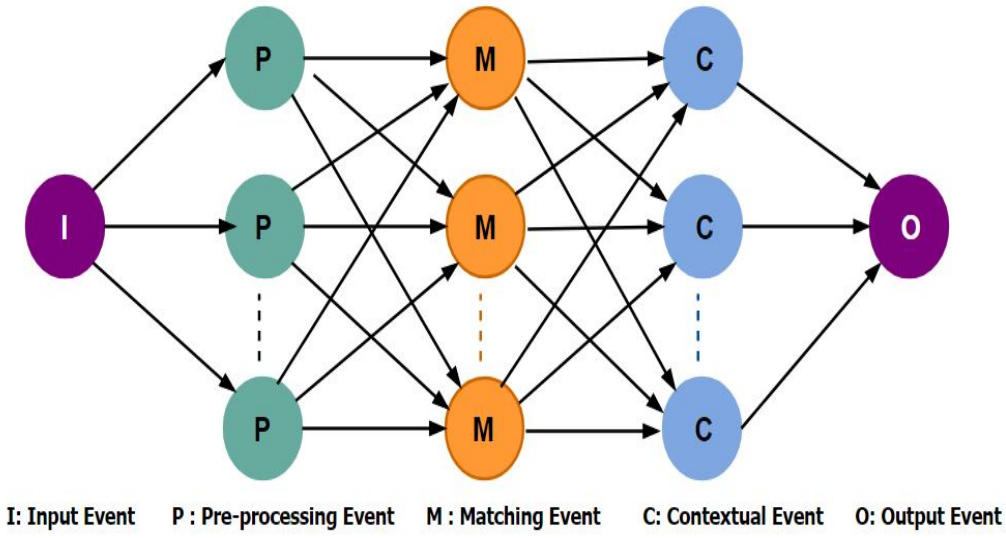


Figure 4.2: Distributed Contextual Anomaly Detection (DCAD) based on DAG model; Input (I), Pre-process (P), Matching Events (M), Contextual Anomaly (C), Output (O).

Data subset division and task parallelisms are the two major solutions to handle scalability concern in the field of machine learning and big data distributed data processing. Importantly, distributed big data stream processing such as Storm is one of the appropriate solution to handle both data and task parallelism in a form of DAG data structure model. The main advantage of Storm is it capability of processing over one million of data streams per second and it capability of fault-tolerance communication discovery from failure. Thus, *event* streams have been designed to be partitioned into a number window to control the high rate of the streams based on one of the most reliable window partition method and according to the most appropriate task paralysis of *FieldGrouping* (). On the one hand, *event* streams can be divided based on their *tuple* value and according to their similar field values. For

example, *tuples* from same value can be emitted to the same *Bolt* in a round-robin approach. This can be achieved by grouping *event* streams from (S_1 = (e_1, e_2, e_3, , e_4, e_5) at between (7:00am to 9:00am) to the first window partition, and second partition window to next constructed window partition at between (9:00am to 11:00am) respectively.

Since high volumes of *event* streams arrive from distributed messaging system, the main task of DCAD is to organise the sequence of *event* streams into an appropriate format of tuples. For example, in Storm, *Spout* is known as first entry point of the distributed stream processing and the main tasks of *Spout* is to pull/ read *event* streams from the distributed messaging system such as Kafka *topic* partitions, converts data source into a stream *tuple*, and then emits them into the *Bolts* within the *topology*. The concept of *pull-based* is to pull *event* streams from the queue in the *topic* partition for further processing. The benefit of *pull-based* approach is to prevent *Bolts* form over flown of *event* streams and guarantees that each *Bolts* are capable process such unbounded sequence of *event* streams.

In the DCAD, two Spouts (Sp_1, Sp_2) are designed and constructed as *Event Stream Spout Splitter* (E3S) to aggregated the input *event* stream. The main tasks of these *Spouts* are to read *event* streams from publisher (producer), converting these streams into a designed *tuple* format of ⟨ *e, t, d* ⟩, and splitting converted *event* streams into across of different *Bolts* within each window partitions (See Section 3.3.2). For example, to handle high input *event* stream volumes (e.g., 1,000,000 *tuples* per second), more than one parallel *Spout* is required in order to control the rate of *event* streams. Then a sequence of *tuples* can be pre-processed or filtered by the designed *Bolts* in the first module. Thus, in DCAD (B_1 to B_4) are constructed for the pre-processing and filtering task.

The main task of each *Bolt* is to provide all the processing functionalities in DCAD *topology* including: filtering, aggregating, joining, writes, read and access to the database. The size of receiving *event* streams from *Spouts* are usually very large, and due to the need of further processing and detection,

multiple *Bolts* are required to be designed and constructed in order to handle such high throughput of *event* streams in real time. The benefit of deploying multiple Bolts is to handle the size of event streams and deploys a different computational function. Thus, in the DCAD, *Bolts* are designed for pre-process *event* streams and divide them into several window partitions based on the two proposed windowing algorithms of *time-based* or *count-based*. The matching *Bolts* (B_5 to B_7) are computes each *event* stream tuple values to detect changes from coming *events* and grouping them according to their rule sets as described in algorithm 3. Finally, the anomaly scoring *Bolts* (B_7 to B_9) computes candidates of *Contextual Anomaly* ($\boldsymbol{C_A}$) from the sequence of *event* stream *tuples* (See algorithm 3, lines 11-26).

Figure 4.3 illustrates an overview of distributed *event* stream processing and detection methods based on *Pre-processing*, *Event Matching*, and *Contextual Detection.* An overview of each module has been described in the following section.

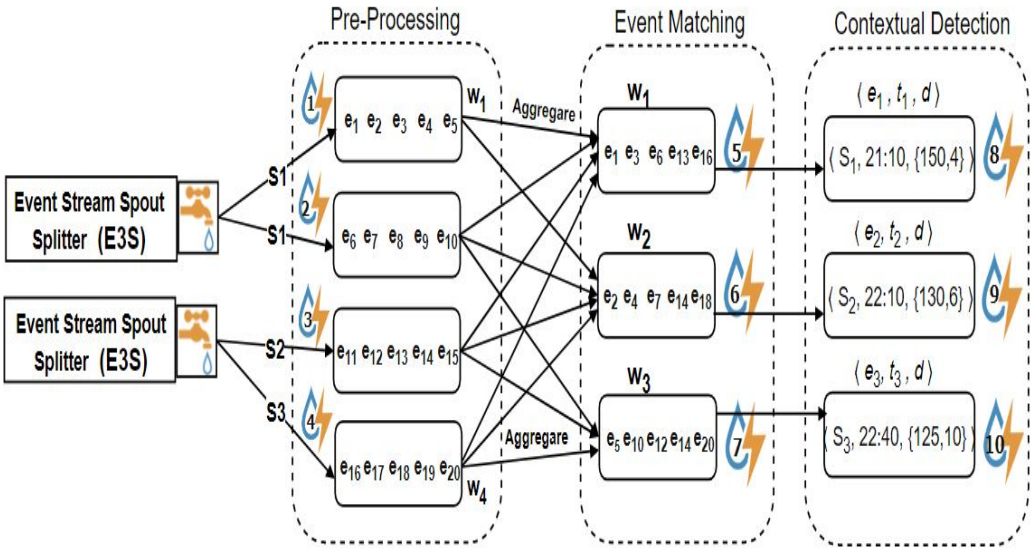


Figure 4.3: Distributed Contextual Anomaly Detection method.

4.2.1. Pre-processing Module

In this module, a window partition is required to be defined before the parallel aggregation is deployed in the *topology*. In every distributed stream processing including Storm, the stream tasks are processing by number of so called operators; the main tasks of these operators are to direct data streams into the worker for execution. These operators in Storm are acting as *Spouts* or *Bolts* to perform different computations over the *event* streams. One of the *pre-processing* tasks is to read *event* streams by (E3S) and transform allocated *event* streams (e.g., *event* name, *event* number, and values) across each window partitions. For example, consider grouping *event* streams according to temporal length of window partitions from w_1- w_2 as shown in figure 4.3. The other task of pre-processing is to filter the *event* streams based on satisfied of each rule condition by stream *tuple* records in each window partitions. The rule condition is based on the *count* and *sum* functions in the windows operators. Consider high and low speed of vehicles (e.g., > 120 km/h or < 60 km/h) within each window partition. A temporal window length can be constructed based on the *event* stream query statement, which is stored in the Redis memory storage as;

*Query 1 (): Speed/**S_1**, [S_i = 'event'] > " ...", Query 2 (): Speed/**S_1**, [S_i > 120], Window/Time (120) > " ...")*

While query1 filters the *event* streams based on the name attribute *tuple*, and query2 filters the *event* streams according to the speed *event* value of speed in each stream *tuples*.

Filter Bolts: The task of each filtering *Bolt* is to filter arrived *event* stream based on *event* satisfied condition rule per each case studies of (e.g., high and low *events temperature)* and (e.g., over or low *speed* of vehicles *events*). For example, a primarily key-value records in each *tuple* depends on a rule condition as defined in each *event* stream *tuple* based on average speed (e.g., *speed* >120km/h). For example, sequences of event streams continuously

arrives for further processing and detection, and consider constructing of window partition over the *event* stream according to high or low tuple values as they can be seen in figure 4.4.

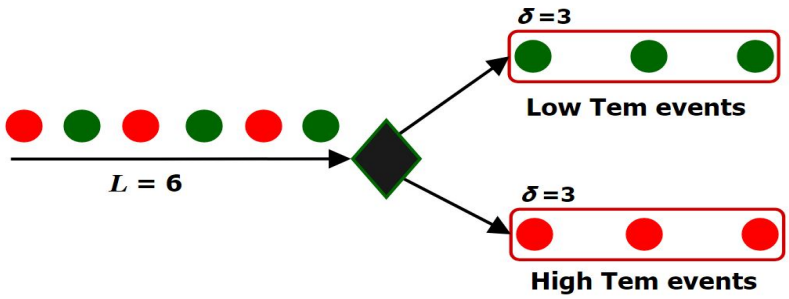


Figure 4.4: Filter event stream per window partitioning based on (e.g., high or low) tuple values.

Figure 4.5 shown the filtered of *events* from e_1 to e_{20} from the number of sensor S_i at a specific time period, then *event* streams are being filtered according to their rule set (e.g., over or low speeds of vehicles) and partitioning them across a number of *n* windows. The filtered *event* streams then can be emitted to the event matching and next *Contextual* detection modules for extracting the *Contextual* behaviour of *event* streams.

Rule-based Bolts: The task of these *Bolt*s is to compute requested *event* streams from filtered *events* and to provide values output, for example, to compute an average speed of vehicles per each window partitions according to the *FieldsGrouping* () mechanism for every two-hour time interval (e.g., 7:00am to 9:00am) from each window partitions, w_1 and w_2 respectively. This improves and reduces a high number of changes in the *event* streams and increases the probability rates of anomalous *event* score. The result then can be tested with arrived and pervious *events* in each window partitions, which have been stored in the Redis[14] memory.

[14] https://redis.io/

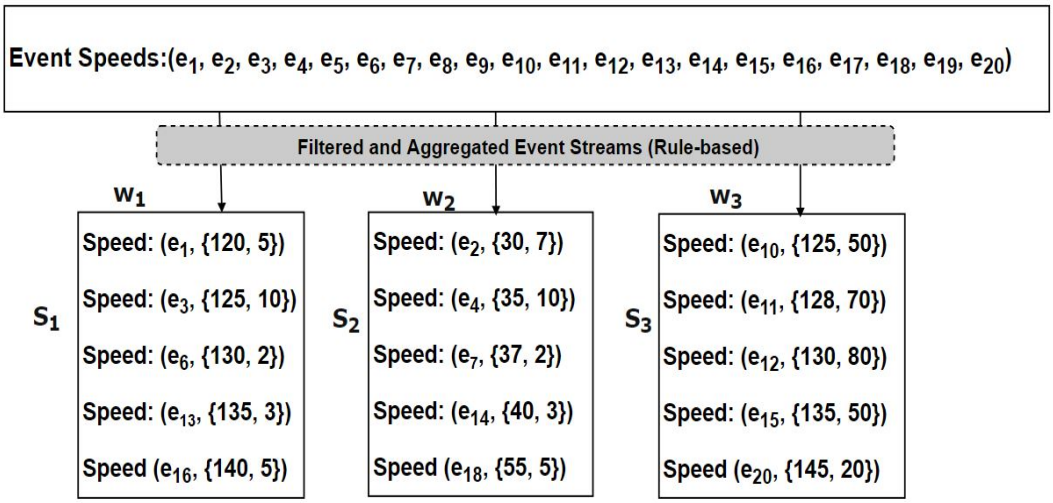


Figure 4.5: Filter and aggregate event stream results from window partitioning.

4.2.2. Event Matching Module

Despite to the fact that *event* stream matching is an important step in the DCAD and *Event* stream *tuples* can be distributed across different *Bolts*. The task of each matching *bolt* is to group *event* streams regularly according to their *tuple* values in each window partitions. For example, w_1 consists of sequence of *event* streams with similar speed *tuple* values according to their rule-set. In this module, matching individual *event* based on the result of window partitions length in the *pre-processing* phase is designed. The main task of matching procedure is to compare the *event* streams in each window partition in order to detect any changes according to their aggregation functions *f* from pre-processed module. As depicted in figure 4.6, *events* from a number of corresponding streams $\boldsymbol{S}1$, $\boldsymbol{S}2$, and $\boldsymbol{S}3$ have been emitted by E3S and partitioned into two aggregation functions $f1$ and $f2$ streams according to their matched results as achieved from their field grouping mechanism. This can be achieved by implementing logical grouping function in the Storm *topology* as *event* stream *tuples* from $\boldsymbol{S}1$ can be aggregated into $f1$, similarly, grouping *event* streams from $\boldsymbol{S}2$, and $\boldsymbol{S}3$ can be aggregated into $f2$.

The benefit of this technique is to match a window partitioning pf each *event* stream results to detect change within the *event* streams and to predict the behaviour of the *event* streams as described in Algorithm 3. The results of matched *event* streams are merged into the *Contextual* anomaly detection module to predict the *Contextual* behaviours of each *event* according to each *tuple* value records using grouping mechanism.

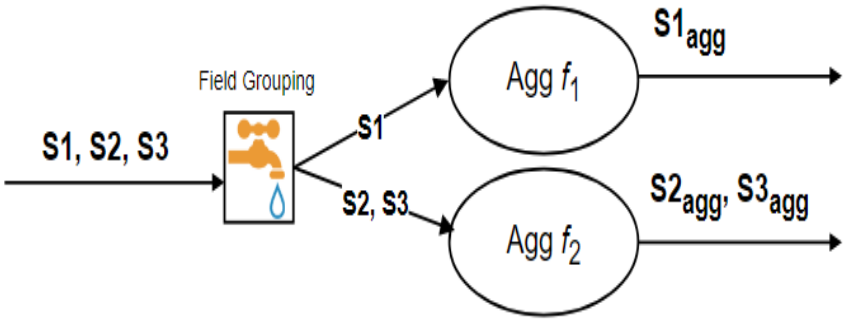


Figure 4.6: Event stream aggregation based on filed grouping mechanism.

4.2.3. Contextual Detection Module

The main task of this module is to detect a number of *Contextual* anomalous *events* from forwarded pre-processed and matched *event* streams. The *Contextual* behaviour is based on *AScore* model (See Section 3.5 definition 8) according to the defined *event* stream in the sequence list with their timestamp. For example, the proposed *Contextual* anomaly behaviour is defined based on the context of *event* in the window sequence with similar *Contextual* behaviour as described in Section 3.5.

The motivation of *Contextual* anomalous *event* detection is derived from aggregating and processing high volumes of sensor streams. This can be achived by grouping e*vents* and detecting such *events* over Distributed Stream Processing (DSP). The ideal solution is to aggregate high volumes of stream

records from multi vehicles of $S = \{ S_1, S_2, S_3 \}$ and partitions streams across distributed computer nodes. Processing data streams in real time and aggregating the final sum of the arrived event streams in parallel within each window partitioning can achieve this.

Consider arriving *events* from number of sensor streams $\boldsymbol{S} = \{\boldsymbol{S_1}, \boldsymbol{S_2}, \boldsymbol{S_3}, \ldots, \boldsymbol{S_n}\}$, where each S_i comprises a number of *event* streams $(e_1, e_2, e_3, \ldots.)$. The aggregation function can be designed based on the sum of the computational *events* to fulfil the condition of Function *F*. The aggregation *F* can be implemented within master-slave distributed processing model for f_1, and f_2 based on DSPE *Grouping* () mechanism. Thus, *event* stream *tuples* belonging to any of S_i can be grouped according to their *event* stream tuple in the DSPE in order to computes and extract *Contextual* behaviour from each window partitions during the aggregation process; for example, *events* from S_1, can be grouped into f_1, and S_2, S_3 into f_2 respectively as shown in figure 4.6.

Example 1: Consider *n* number of stream *events* from $\boldsymbol{S}_1$ to $\boldsymbol{S}_3$, which are aggregated, based on two-hour time interval t_i. As showing in figure 4.6 green circles are associating with the number of *event* streams, for example, from traffic point of view as lower speeds of < 60 km/h, and red circles are representing a number of *event* streams from S_1 to S_3 for over speed of > 120 km/h. As the *event* streams have been aggregated, similar behaviour according to their time interval can be matched according to their *Contextual* behaviour (e.g., *event* speed *tuple* records). Assume $\boldsymbol{S}_1$ consists of several over speed *events* between t_2 and t_3 and similar speed *events* are repeated in t_5; thus, such *events* are can be considered as *Contextual* anomalous *event*. A detailed description of *Contextual* behaviour is described in Section 3.5. Similar speed *event* behaviour is repeatedly occurred in $\boldsymbol{S}_2$ and $\boldsymbol{S}_3$ at different time intervals.

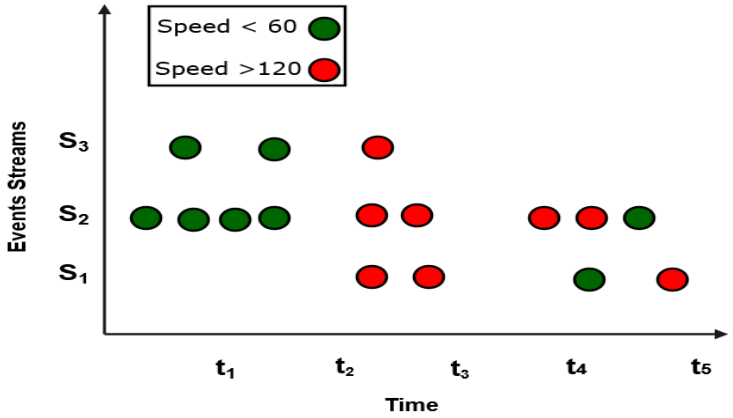


Figure 4.7: Event stream aggregation from set of sensors according their time-based.

AScore Bolts: The main tasks of these *Bolts* are to compute *Contextual* Anomaly $\boldsymbol{C_A}$ model according to their *Contextual* behaviours in algorithm 3. As *event* streams have been pre-processed and matched according to their rules set which is defined in algorithm 3 (CESA), then CESA computes the output result rates based on the *AScore*. This can be achieved by assigning [0,1] scoring values into each *event* streams based on their value in each *tuple*. Consider vehicles *event* speeds of 30km/h at 8:00 am as normal behaviour due to the traffic congestion in the morning; on the contrary similar speed at midnight (12am) can be considered as anomalous *event* This can be due to the unexpected incident on the highway when the traffic speed slows down, in this situation the *tuple* value records as *event* is changing. Therefore, the scoring technique can be designed based on 0 for normal hour for every two-hour time interval such as between 10:00-12:00 and 1 as anomalous *event* at between 17:00 -19:00.

The process of anomalous *event* detection in this stage is associated with times of *event* streams occurrence in the *event* sequence based on predicting *Contextual events* within each window partitions. In this stage, algorithm 3 computes the probability of the *event* score from each partition window. The result of *event* stream from *event* matching module can be computed by the

CESA algorithm 3, *Contextual* anomalous *events* based on the number of *events* in the sequence can be represented and according to their temporal tuple behaviour (refer to figure 4.7). For example, first *event* at 7:45 in S_1 is considered as *Contextual* anomalous; this is based on the computational behaviour according to the scored tuples within the *event* sequence. Specifically, this can be computed according to their similar behaviour that for example repeatedly occurred at 22:00.

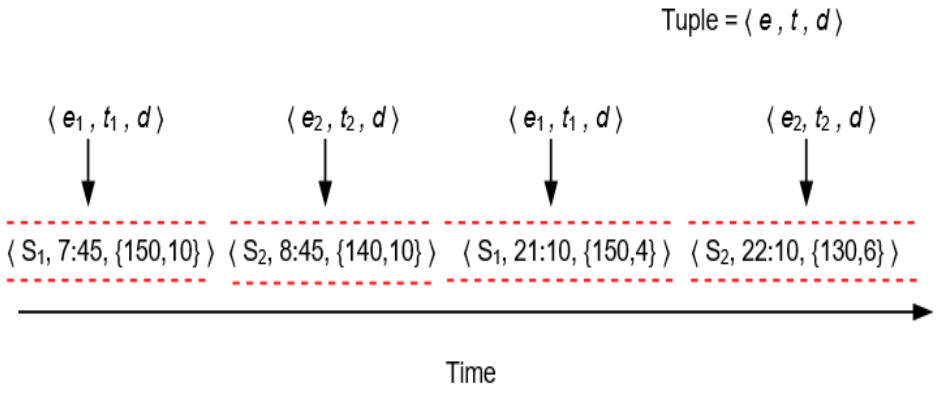


Figure 4.8: Contextual anomalous event detection over sequence of event stream tuples.

Chapter 5

5.1. Experimental Environment, Evaluation Metrics and Result Discussion

This chapter describes the experimental environment, results and performance evaluation for the proposed Algorithms and methods. Section 5.2, describes the experimental environment and settings according to the need of distributed computing cluster to process high throughput of unbounded sequence of *event* streams. Section 5.3, describes the results of distributed *event* stream partitions and pre-processing results. Section 5.4, presents the experimental evaluations for high volumes of *event* streams. Section 5.5 evaluates the experimental results from the proposed algorithm in relation to *Contextual* anomalous *event* results using a scoring rate. Section 5.6, describes the predictive accuracy and error rates of the algorithm. Section 5.7, presents the results of change detection in the *event* streams. Section 5.8, presents the CESA algorithm computational complexity and performance. Section 5.9, presents the results of DCAD framework performance in relation to the scalability of the proposed modules in real time and the variant effects that possibly have an impact on the performance in processing time and low-latency response. The summary of the chapter is described in 5.10.

5.2. Experimental Preliminaries

As high volumes of unbounded sequence of *event* streams arriving at a very high speed, continuous online learning is crucial to evaluate the effectiveness of the proposed CESA algorithm. In that context, the result is mainly based on the following challenging objectives.

1. Detect *n* number of *Contextual* anomalous *event* behaviour based on the proposed $\boldsymbol{C_A}$ model and achieved *AScore* output result rate per every distributed partition window length of (e.g., 100,000, up to 1 million) *event* stream tuples respectively.
2. Evaluate the DCAD performance based on the computational accuracy and memory consumption.
3. Prediction error rate; lower error rate indicates the higher accuracy of the algorithm in relation to detecting change in the *event* streams.
4. The cause of change detection in each of the window partitioning according to the *PH* Test computational results.
5. The cause of the algorithm's performance processing time according to increasing and decreasing number of window partitions.

5.3. Experimental Environment

The proposed algorithms are implemented in Java programming language as part of the DAG *topology*. The experiments were run on the University of Huddersfield distributed High Performance Computer (HPC) cluster of eight computer nodes. The cluster consists of one master node known as (nimbus) and seven supervisor nodes, where each computer node is equipped with 8GB of RAM, configured with an Intel(R) Core(TM) 4 Quad CPU Q8400. All the nodes are run on Ubuntu Xenial (v16.04.1 LTS) operating system with

deployed Java(TM) SE Runtime Environment (build 1.8.0_10), and Java HotSpot (TM) 64-Bit Server VM. Each proposed computer node is divided into 4 workers according to the designed distributed communication channel Zookeeper Server (ZkServer) as illustrated in the DCAD architecture in Appendix 3. The task of ZkServer is to create an efficient and dynamic coordination between each node and provide fault-tolerant service. Finally, several *topologies* are created to be deployed across the cluster in parallel. The main job of each *topology* is to assign tasks between each DCAD's module and managing the task scripts distribution between the nodes.

5.4. Data Sources

One of the major concerns in any anomaly detection method is to propose a right data type and format to computes the anomalous results, when most of the data is formatted and collected differently. In this thesis, two IoT case study data sources are used in a form of stream to support the research outcome result and fit to the distributed data structure methods. For example, monitoring smart traffic on highways and detecting high speed or congestion speed *events* is playing a signification role in terms of safety, reducing congestion, and saving drivers time to alter their journey. Consider a global speed limit on highways roads as 120km/h. Thus, over or higher speed of (e.g., 140km/h) or lower speed (40km/h) can be considered as unusual *events* or activities. Importantly, in this thesis, such behaviour is considered as anomalous *event* as described and defined in Chapter 2 and 3. The first data size is comprised of 210 million streams instances and 8 attributes. The summary of the data schema is described in Table 5.1.

Attribute	**Description**
Sensors ID	Identifier of the network Init sensor
Date and Time	Date and Time of vehicles speed measurement
Average speed	An average speed per vehicle (km/h)

Flow	Count *n* vehicles with average speeds
Headway	Average time between vehicles
Occupancy	Occupancy of loops (%)
Travel time	Average time to traverse the section (s)
Prof travel time	Expected travel time based on historic profiles

Table 5-1: Highway road traffic data schema.

The second proposed data source to detect anomalous *events* from was a remote temperature stream. This data source is collected from Intel Berkeley Research Lab IBRL[15]. The sensor network is comprised of 54 sensors with different data attributes of *temperature*, *humidity*, *light*, and *voltage*. The sensor data are collected through Kafka Application Programming Interface (API) via implementing *topic-based* method. These data sources are then distributed through *publish-subscribe* messaging system as depicted in figure 4.1. The sensor data is converted into a stream *tuple* format based on the distributed stream processing data structure model. In this context, processing high volumes of *event* stream was a challenging task. As a first challenge, proposed number of data attribute is required as prior assumption of the time-series format. Thus, first, only *temperature* data attributes are collected from the *topic* partition of e.g., *temperature* from S_i stream to create *event* stream *tuple* from. A second challenge was to propose different data sources to test and train the proposed algorithm and CA model to detect anomalous *Contextual events* from such sequence of unbounded of *event* streams.

As described in the previous section, both proposed IoT data sources are consisting of different data attributed, which are irrelevant to the anomalous *event* detection. Thus, several pre-processing tasks are made including data cleaning, and data transformation before the *event* streams can be emitted to CESA algorithm for the learning process. First, for the constancy of the detection purpose, stream sensors have been cleaned from missing values, removing *tuples* with zero records. Second, sensors values have been

[15] http://db.csail.mit.edu/labdata/labdata.html

transformed into *tuple* stream format by E3S *Spouts* to be fitted into the distributed stream processing data structure model. Such task is performed by one of the DCAD *pre-processing* modules and the size of sensor streams have been filtered based on the rule set of anomalous *event* stream *tuple* records; for example, only *event* stream *tuple* with either high or low records is emitted in the next module. The main advantage of this approach is to reduce the size of *event* streams from irrelevant attributes and dividing them into an appropriate format for the window partition algorithms to handle. Thus, sensor streams from only ($\boldsymbol{S}_1$, $\boldsymbol{S}_2$, and $\boldsymbol{S}_3$) is mapped and filtered based on the *fieldGrouping* (e.g., *temperature*) and *event tuple* values are reduced into approximately 1,0335,000 from over 2.3 million sensor readings. The *temperature event* streams are then normalized based on the *minimum* and *maximum tuple* values of (e.g., 20 °C and 26 °C) before these *event* streams are emitted for the *Contextual* detection. *Event* streams are also normalised into [0,1] based on there high and low *temperature* ranges. Where 0 referred to all anomalous *events* during the normal hours of (e.g., 7:00AM) and 1 to as anomalous *events* at, 23:00PM) Additionally, the arriving new *event* stream is defined based on their *tuple* records of (new_min_e) and (new_max_e) and computed in $e_i = \frac{\text{e}-min_e}{max_e - min_e}(new_{max_e} - new_{min_e})$.

5.5. Results and Discussion

5.5.1. Distributed Event Stream Window Partitioning Results

This section describes the performance of the proposed distributed windows partitions algorithms to partition *event* streams into several window w_p based on either counted *tuple* or time interval observation. The result of count-based window algorithm's computational is presented in figure 5.1. The algorithm is constructed based on the number of equal lengths of window partition and n = 100,000 *tuples* in very L. Green Δ symbols are indicating of the lower *temperature event* streams; on the contrary, red × symbols are indicating of the high *temperature event* streams in every window partition. The size of each tumble window partitions is tested on average δ = 10 over one million *event* stream tuples. The computational for the high and low *temperature event* is based on the tested and computational results, which have been made by each bolt in the Storm *topology*.

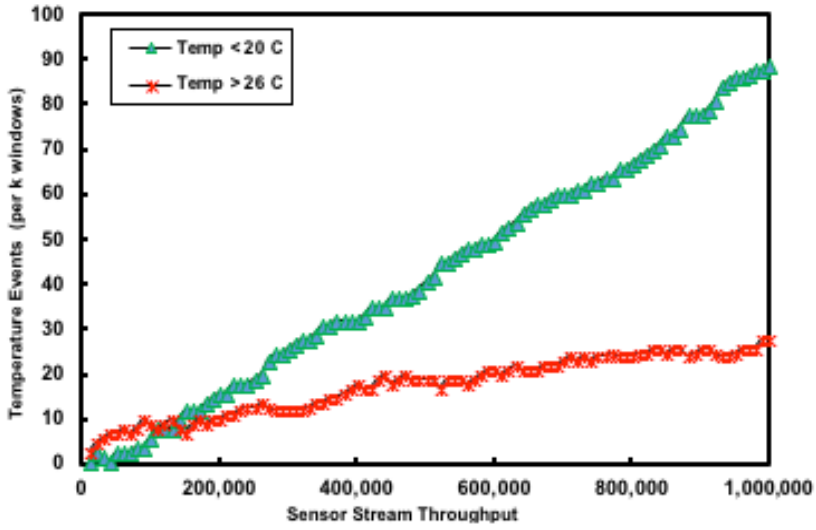


Figure 5.1: Event-based window partitioning result for temperature case study.

A detailed of *count-based* window partition algorithm results of high and low *temperature event* streams are described in Chapter 3. The algorithm is first checked for the constructed windows w_p per *n tuples* in every *L* length size to test the accuracy of captured *event* streams. The result is indicating that the numbers of low temperature event stream tuples (e.g., ei < 20 °C) are much higher than the high *temperature event* stream *tuples* (e.g., e_i > 26°C). For example, 3 *events* with high *temperature* values are captured in w_1 over the length *L* = 100,000 *event* streams *tuples*; thus, such *event tuples* are considered as anomalous *event* in the data stream sequence according to the predefined notation as described in Chapter 3. Similarly, 19 anomalous *events* are captured when *L* = 500,000 stream *tuples* based on the factor of δ = 5. In this experiment, only *high* and *low temperature events* are considered to evaluate the performance of window partition algorithm due to the correlation between these attributes *Contextual* behaviour. Importantly, our approach is developed to detect several *Contextual* anomalous *events* over high volumes of sensor streams in parallel.

The complexity of expanding window length has a signification impact on the high accuracy of the *event* score results. This indicates that smaller window partition length is requires less memory space and efficient computational result. In this context, count-based window partition algorithm is mainly depending on the counting *events* statues (e.g., high or low) per each window length *L* until the size of *w* is satisfied.

In the second data source scenario, only *speed* attributes and the number of *event* flown are most relevant data types to discover the *Contextual* anomalous *events* from. Detecting *Contextual* anomalous from high volumes of *speed events* plays an important role to predict the state of highway road traffic in real time. Thus, we *pre-processed* and aggregated *speed events* based on their sensors values and defined attributed of *high max* () or *low min* () as defined by equation 3.12 in Chapter 3. Figure 5.2 depicts the result of Algorithm 1, where *event* streams are partitioned and grouped according to their *high* (e.g., 120km/h)_or *low* (e.g., 60km/h) *speed event*s. The *event* streams are aggregated

based on *n* number of *speeds event* stream tuples for every L = 100,000 per each window partition w_p as labeled in *a-axis.* The proposed algorithm is tested on δ = 10 over one million *event* stream tuples. The result is demonstrated that the numbers of over speed events are much higher than the lower *speed events* in each window partitions. The result of *event* stream computational is more reliable across the experimental learning of *event* detection.

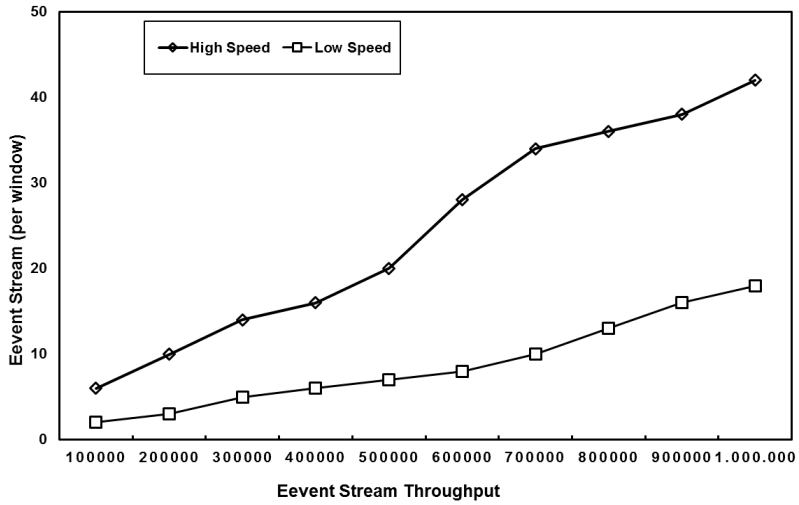


Figure 5.2: Event-based window partitions result from traffic case study.

Additionally, a detailed description of the result is illustrated in Table 5.2. Window partitioning columns are referred to the *n* constructed number of window partitions, while *Stream* columns are associated with *n* number of *event* stream per window partition, and the last columns are repenting both *high* and *low* of *event speeds* per w = 100,000. For example, in w_1 9 *event* stream *tuples* are captured with high records and 2 *event* streams *tuples* with *low speed* records. The learning computation is sequentially repeated for every *tuples* of stream across each window partitions until the window length L is satisfied.

Window	Total *Event* Streams (w_p = 10k)	High Speed *Events* (per w_p)	Low Speed *Events* (per w_p)
w_1	100000	9	2
w_2	200000	22	2
w_1	300000	24	11
w_2	400000	32	2
w_1	500000	34	4
w_2	600000	32	8
w_1	700000	38	12
w_2	800000	42	13
w_1	900000	54	18
w_2	1.000.000	58	13

Table 5-2: Result of event streams per window partitions from road traffic case study.

The result of Algorithm 2 *time-based* window partitioning is presented in figure 5.3. The input *event speed* behaviours are collected according to the time of the speed occurrences, where for example*, y-axis* represents aggregated *event* streams at between 7:00AM to 21:00PM hours intervals metrics. Thus, the computational result is based on *high* and *low* speed features behaviours in every S_i. Where, *x-axis* represents a number of *high* and *low speed events tuples* per window partitioning w_p. The *high* speed of *event* is represented in (red Δ) symbols and low speed of *event* (green ×) symbols. The results from window partitions are indicating that the number of low *event* speeds on average at every t = 2 hours interval is significantly higher than the higher *event* speeds. The lower *event* speed is approximately 50% higher than the high *speed events* across the learning process. Organising these computational results is significantly important for the next objective of this thesis to detect *Contextual* behaviour from the *event* streams within every window partitioning.

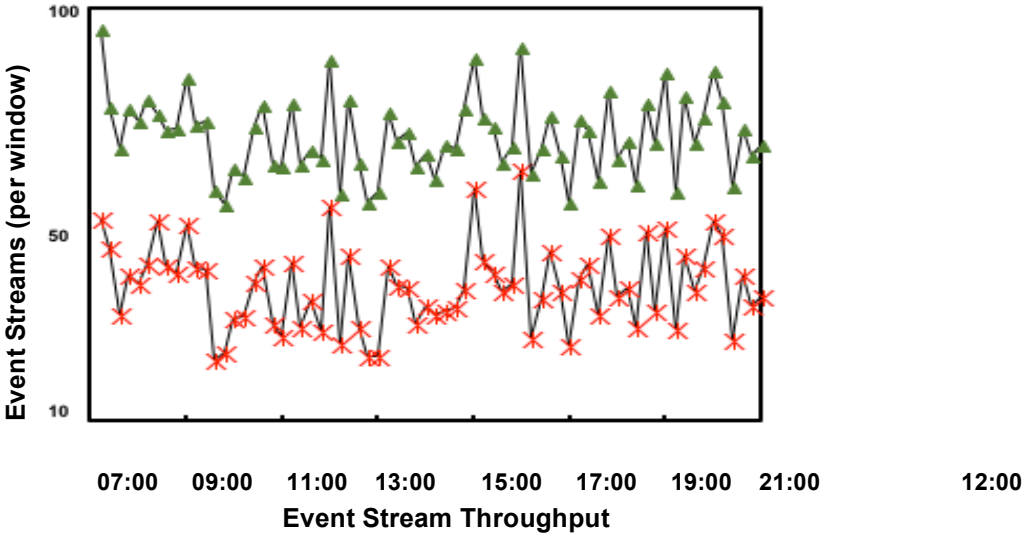


Figure 5.3: Time-based event stream window partitions results from highway road traffic sensors.

5.5.2. **Contextual Event Stream Anomaly Result**

Modeling *event* stream is one of the most complex and challenging tasks to detect the *Contextual* anomalous behaviours. Most of the existing anomaly detecting output results are defined based on two criteria; a) labeled normal and abnormal data, or b) assigned scores to the output result. In contrast to statistic dataset, detecting anomaly results from sequence of *event* stream on online poses a significant challenging task. In this thesis, the second challenging scoring criteria are considered as appropriate solution to evaluate the output scoring results of Contextual behaviour rates. The CESA algorithm is computed based on the probability of *Contextual* anomalous *events* scoring result and according to the Equation 3.6, 3.7 and 3.8. The $\boldsymbol{C_A}$ computes the *event* streams according to the *event* occurrence time based on their *Contextual* behaviour within each window partition. Thus, CESA algorithm is implemented to detect *Contextual* behaviour from *event* streams after they have been aggregated and pre-processed from the previous window partitioning algorithms. The algorithm is learnt and trained from 100,000 *event* stream *tuples* to evaluate the proposed $\boldsymbol{C_A}$ model. This is approximately 10% of the completed *event* stream size and

the rest of 90% *event* stream *tuples* are proposed to test the probability of *AScore*. The algorithm's learning procedure depends on the shuffling *event* streams into grouping mechanism from *every* window partition to measure estimation error or false negative rate as described in Section 5.6.

As shown in figure 5.4, the *top-axis* is represents the size of *event* stream from traffic sensors to train the C_A model. Where, *x-axis* is representing a computed scored of *Contextual* results based on sliding factor $\delta = 3$ interval metrics at between [t = 0:00, t = 12, t =24:00]. The time interval metric is mainly according to the result of detected *event* by the time-based window partitions algorithm. This is achieved by computing *Contextual event* behaviours according to high or low *tuple* of speed records as normalised in [0,1]. The model is then trained across every *event* partition of $w_p = 200{,}000$ *tuples*. On the other hand, the *y-axis* is representing the number of high speed scores in (red Δ symbol), in contrast to the normal *events* in (green Δ symbol).

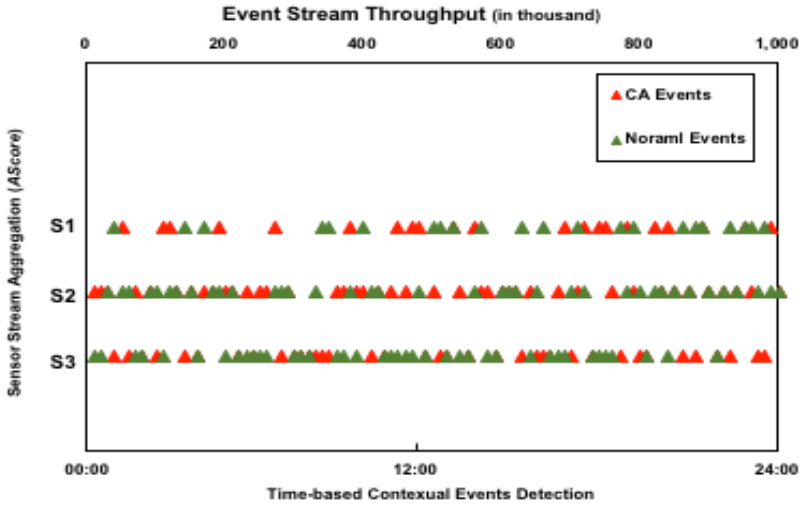


Figure 5.4: Contextual event stream result from highway road traffic streams.

The results of CESA algorithm based on observing aggregated number of *f* functions from sequence of sensor streams in parallel is shown in *x-axis* in the figure 5.5 while *y-axis* is representing the result *Contextual* behaviour. In this

context, the result is demonstrated that ***S***1 is consisted a higher number of *Contextual* behaviour compared to ***S***2 and ***S***3 per each window partition. This is due to the smaller number of *events* in each window partitions, when each of *event tuples* is observed and matched according to their *Contextual AScore* after they all *event* streams are mapped according to their *Contextual* behaviours based on w$p\ (e \pm Si) = e\frac{1}{2}(\alpha \pm \beta)\, Si\frac{1}{2}(\alpha \mp \beta) > 1$. This is indicating that on average, 50% of aggregated *Contextual* behaviour are occurs in ***S***1 rather than other two set of sensor streams ***S***2 and ***S***3. The number of *Contextual* behaviour per window is increased linearly as time progressed in addition to scaled up *n* number of *event* streams. Thus, it can be argued that the *Contextual* behaviour is not only based on the context of the *event* streams, as it can be depending on the number of *events* streams per window partitions.

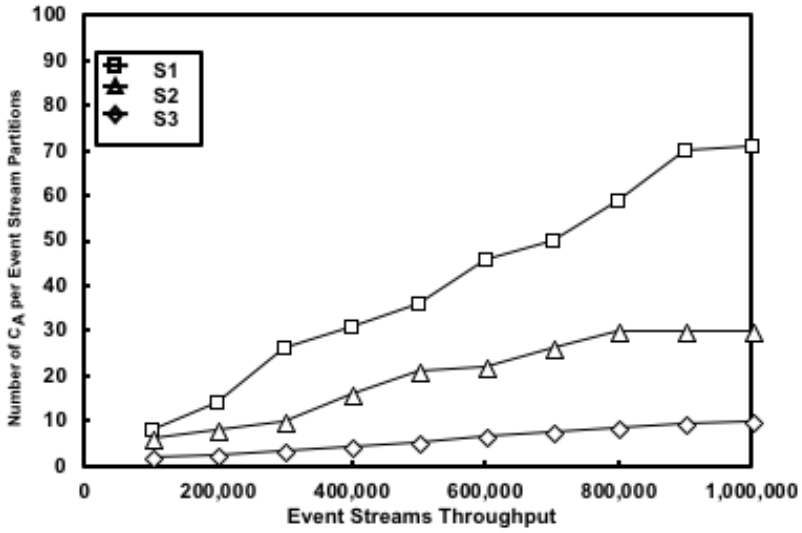


Figure 5.5: Contextual anomalous event detection from aggregated and matched events per window partition over variant streams sensor devices.

5.6. CESA Algorithm Performance

The evaluation of CESA algorithm's performance is based on several facts as follows;

i. *AScore* accuracy rate

ii. Size of *event* streams.

One of the most important aspects of the proposed CESA algorithm is the capability of the algorithm to learn to detect *Contextual* behaviour on online at the same speed of the *event* streams. The experimental evaluation of CESA algorithm is mainly depends on the output result of *AScore* and the size of *event* streams. The results of CESA performance to detect *Contextual* behaviour from the proposed data sources *event* streams are depicts in Appendix 6, figure 5.10, and figure 5.5. The result of *Contextual* anomalous detection algorithm is achieved based on *n event* streams with O(*n*) processing time. The probability of *AScore* computational for each *event* stream *tuple* is predefined as [0,1], where high scoring is referred to high probability of *AScore* for the true *event* rate $e > 0.5$; thus, such result is refers to a true positive scoring result and stability of the algorithm. The probability of negative rate for each *event* stream is achieved as less than $e < 0.5$ scoring rate.

The result has indicated that the model is learnt from the *event* stream and the high accuracy of positive prediction error rate is acceptable for such size of *n event* streams since *e* is frequently detected in total *m* Since $\sigma\,(e) > \frac{n}{m} > \frac{n}{m+1} > l$, while high accuracy of *AScore* indicates the stability of the CESA for scoring anomalous *events*. Importantly, the output result of scoring over the total size of *event* stream is mainly depends on the learning algorithm after iterating from the change detection. On the other hand, processing a smaller size of *event* streams values per window partition has indicated the less computational complexity in terms of O (k log n) performance. Thus, high throughput of *event* streams is not guaranteed to improve the higher negative

rate, achieving less accurate scoring rates. According to the result in figure 5.10, an alternative approach of proposed partitioned *event* streams across number of windows and deployed CESA algorithm is improved the high rate of positive scoring computation.

To evaluate the performance of CESA algorithm, *Contextual* anomalous *event* streams detecting is implemented according to *AScore* in C_A model for each case study. Thus, the aim of such experimental result is to estimate and evaluate the accuracy of CESA computational degree in both scenarios in relation to anomalous *event* score. The rate of *AScore* in the *event speed* scenario is slightly higher than the rate of *AScore* in the *event temperature* scenario (refer to figure 5.6). The result indicates that the capability of CESA algorithm to compute anomalous output score rates is acceptable since, one of the main criteria to evaluate the anomaly detection method is high probability of the computational scoring rate.

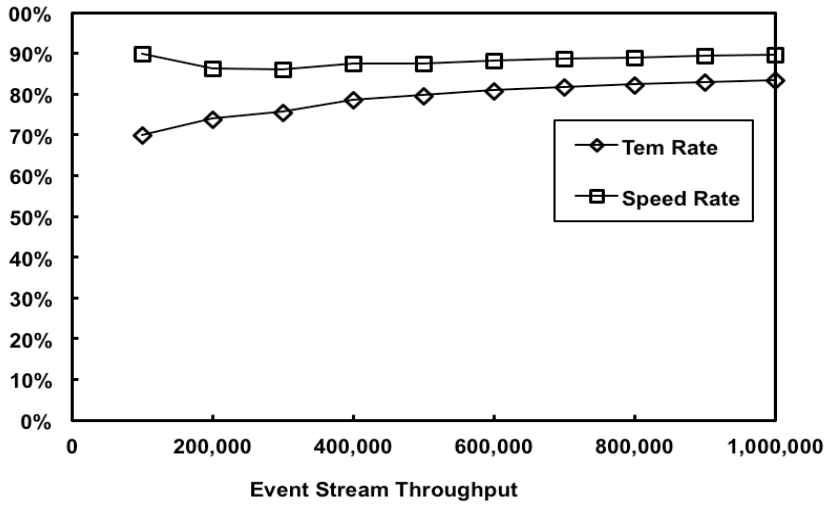


Figure 5.6: Result of CESA algorithm accuracy performance rates over two IoT proposed case studies.

5.7. Contextual Anomaly Detection Scalability Evaluation Result

The size of processing *event* streams can have major impacts on the computational performance with regards to proposed both centralised and distributed computations. In this thesis, we evaluate the scalability detection processes in regarding distributed *Contextual* anomalous detection performance. The evaluation performance result is measured based on;

1. ***Event Stream Size***, the proposed *event* stream size threshold is stetted as t =100,000 stream tuples per node to evaluates the effectiveness of the computational performance.
2. ***Number of window partitions***, which is evaluate to assess the impact of the computational results and execution performance.
3. ***Scalability*,** increasing number of computer node can perform effectively with less processing runtime performance according to Equation 5.1. Where p is referred to processing runtime performance time for every N node with the proposed cluster.

$$p = \frac{{}^{1}N\ processingTime}{processingtime_{Nnode} \times N} \tag{5.1}$$

This is the most common metric to measure the parallelism performance for the runtime detection process.

Importantly, the experiment results evaluation is based on the combination of the above factors. The performance of CESA can be argued according on two major facts, processing time and detection accuracy. First, the impact of scalable *event* streams on processing runtime for both centralised with distributed approached are shown in figure 5.7. The result is indicated the performance of the detection process is linearly increased the size of *event* streams are increased. In addition to this, the result of implementing CESA

algorithm centrally requires more computational detecting processing runtime to computes matching and detecting anomalous *events* compared to the distributed method. In this context, we tested the algorithm with a set size of *event* stream threshold of 100,000 per computer node to evaluate the detecting computational performance.

The processing of the detection is primarily based on the CESA algorithm's runtime performance based on distributed and centrealised methods. The performance of CESA algorithm's processing runtime is recorded in millisecond (ms) as depicted in *y-axis*. It is evident that process and computes *Contextual event* streams over 800,00 *tuples*, so the CESA algorithm is required for less than 400 milliseconds (0.4 ms). On the contrary, for testing similar *event* stream size with threshold $e > 1k$ are expected higher processing runtime of 1120 (1.12 ms) is expected to computes the detection centrally. Importantly, as the size of *event* stream threshold $e > t$ is scaled up to 800,00, the detection process of the runtime performance has also linearly increased and doubled. The result has demonstrated that the proposed CESA algorithm over DCAD framework is performed effectively with regard to the processing of detection performance runtime in real time. As described in Chapter 2, processing runtime and low-latency (real time) are the two major concerns in real time anomaly detection to have impacts on the detection performance and computational results.

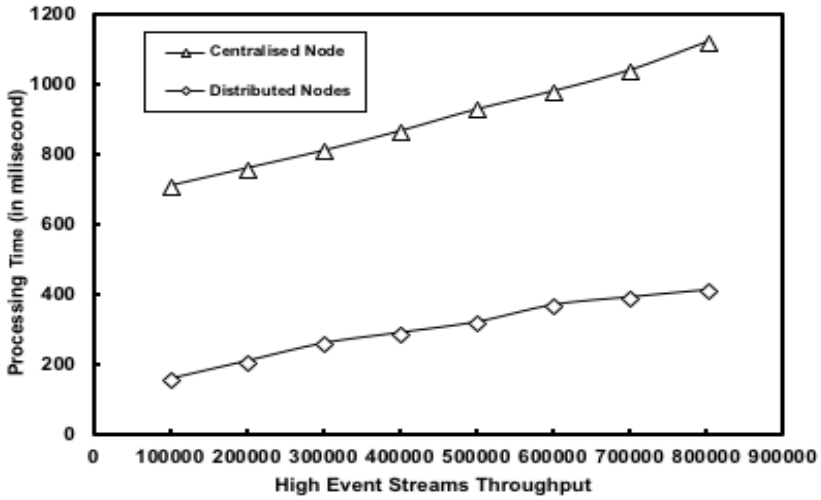


Figure 5.7: CESA processing performance based on standalone node versus distributed nodes with threshold of processing 100k event streams.

In addition, another the key challenging task to process large-scale of *event* streams in real time is to proposed the right number of window *w* and *event* size length *L*. Consequently, a small size of window partition in each computer node can be more reliable regards to *Contextual* anomalous *event* scoring result. On the other hand, large size of proposed window partition is an alternative approach to enhance the detecting runtime performance, since the computational accuracy is the major concern in this approach, specifically, when the size of *event* stream per each window partition is high, this due to the fact that *event* stream evolves, and windows are possibly disjoint. Figure 5.8 depicts the running time per each window partition in milliseconds as presented in *y-axis*, while *x-axis* presents high number of *event* streams throughput.

As the result has shown, using different numbers of window partitions per computer node is reliable in relation to changing detection process. Hence, decreasing the number of windows per computer node improves the processing runtime latency. For example, consider window partition $w = 2$ per single node for the length L= 200,00 *event* stream *tuples* detection process, since this process is approximately required 30 seconds. As the experimental results evidence that when the length of *L event* stream *tuples* is doubled, the

processing runtime is also to increasing. Scaling up the number of *event* stream to, one million *tuples* requires 120 milliseconds to complete the detection process. Importantly, increasing the number of window partitions from $w = 2$ to $w = 4$ per computer node is required 100 milliseconds. The result demonstrates that increasing the number of windows partitions up to $w = 8$ requires less than 75% processing runtime compared to the *event* stream scaling up size. In this context, the main benefit of increasing window partitions in parallel is to reduce the size of memory space on each computer node and improve the computational processing runtime performance as argued in Gama (2013). The current implantation of the proposed window partition algorithm is performed effectively with regards to low latency response and online learning, since these are a two major critical and highly concerns in most of the streaming application domain to control high speed of the *event* streams from disappearing or being disregarded.

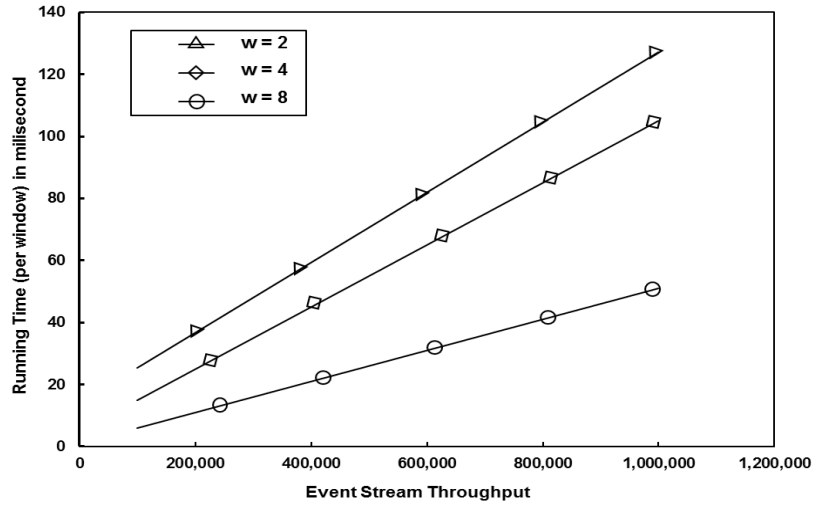


Figure 5.8: Performance of variant window number partitions computational processing time.

One of the key concerns in this thesis is how to handle high volumes of *event* streams in real time, since scalability is playing an important role in terms of low-latency processing runtime response. For example, to measure the

effectiveness of DCAD performance, the result is primarily dependent on the execution runtime per computer node. The proposed approach is based on the number of tasks performed per workers over *n* number of nodes *n* > 1as described in Equation 5.1. The performance of the scalability of the *event* streams detection process result (refer to figure 5.9). The result has demonstrated the effectiveness of parallel processing performance and allocating *event* streams across the proposed cluster. The performance of each computer node is recorded in milliseconds (ms), for example, to process high volumes of *Contextual* anomalous *events* (e.g., 1,000,000) requires more than one computer node in the DCAD framework. In this context, we doubled the proposed nodes from 2 to 4 nodes to computes (e.g., 200,000) *event* streams. The result indicates the computational performance runtime of detection process is improved by 25% as labeled in *y-axis*. On the other hand, a similar size of *event* stream detection process with 6 nodes has significantly improved the computational performance up to 50% of the processing runtime. Overall, the experimental results have indicated that CESA algorithm satisfies both the DCAD computational result and have processing runtime performance for over one million *event* stream *tuples* in less than 1 second. This result is supported and improved based on the capability of distributed Apache Storm framework for processing one million stream *tuples* in 1 second. Note that our evaluation is mainly based on proposed distributed computational matching and *Contextual* processing of detecting *event* streams in real time. Our method is evidence that the runtime of distributed *Contextual* detection has satisfied the scalability of the detection process, since our condition facts are remained stable (e.g., number of window partitions and number of nodes).

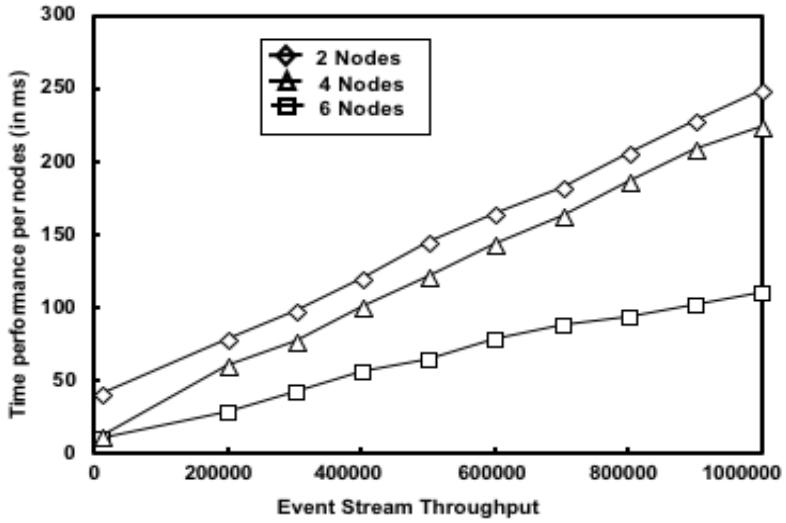


Figure 5.9: Result of increasing nodes linearly performance in the DCAD framework to with scaling up event stream throughput.

5.8. Point and Contextual Anomaly Detection Results

The *event* streams per window partitions are categorised into two correlation coefficient metrics of *Point* and *Contextua*l, with the task of Common Correlated Attribute (CCA) is to compare the value of a pair of *event* stream tuples. In this context, person correlation coefficient is found to be the most widely metric to measure the correlation between *Point* and *Contextual* anomalous *event* streams per window partitions. This is computed as $e\ (x, y) =$ (x,y) .

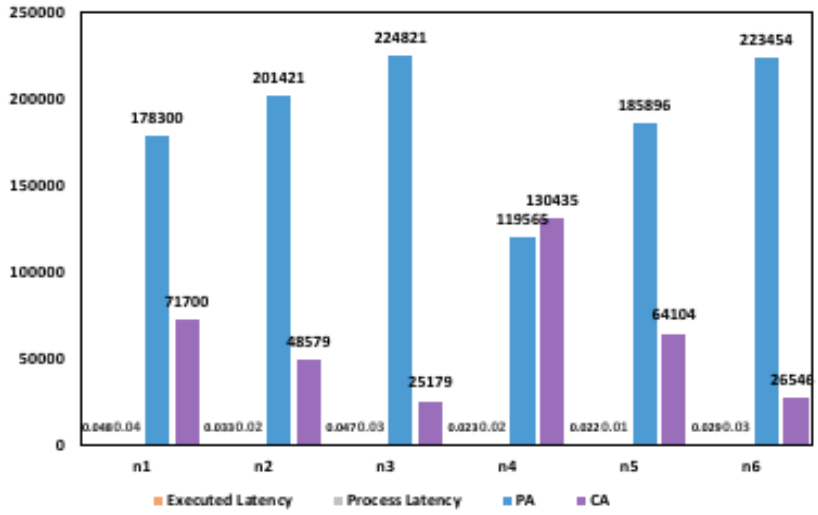


Figure 5.10: The result of distributed Point Anomaly (PA) versus Contextual Anomaly (C_A).

Table 5.3 presents the results of both *point* and *Contextual* anomalous *event* detection tests over 2.5 million of stream tuples. The size of data stream is used after the *event* streams have been pre-processed and matched their tuple values to check, for example, *lower* or *higher event* speed records. The size of *event* streams then has been partitioned across of seven nodes, and each node managed to handle to process 250k *event* stream tuples. The result of each anomaly types is demonstrated in last two columns of the table 5.3. For example, in n_1 both Point or *Contextual* anomalous detection is tested over 250k samples of *event* streams; the number of point anomalous *events* is higher than the *Contextual* anomalous *events* since the completed computational runtime required is only 0.048 milliseconds. Overall, the results of both *point* and *Contextual* anomalous *events* are mainly based on the *AScore* rate and C_A model computational testing metrics per each partition set of tuples.

Allocated Nod	**Event Tuples**	**Executed Latency**	**Process Latency**	**PA**	**CA**
n1	250000	0.048	0.04	178300	71700
n2	250000	0.033	0.02	201421	48579

n3	250000	0.047	0.03	224821	25179
n4	250000	0.023	0.02	119565	130435
n5	250000	0.022	0.01	185896	64104
n6	250000	0.029	0.03	223454	26546

Table 5-3: The comparison *Point* and *Contextual* anomaly computational results per computing node.

Figure 5.11 depicts the CESA algorithm's computational performance in real time, thus, In this context, the DCAD method is evaluated based on the performance of distributed DAG *topology* processing runtime per each for both sensor stream case studies and labeled in *y-axis*. This is evaluated based on the number proposed *event* partition detection computational in parallel in with the *topology*. For example, scaling up the computer nodes in the DCAD framework demonstrates that it can have a major impact on the computational processing runtime as shown in *x-axis*. Thus, adding more nodes into the distributed method indicates more efficient processing runtime across each case studies datasets. For example, to deploy standalone node with a larger size of *event* streams is required over 60 milliseconds to detect anomalous *events* computational results from the road traffic sensor streams scenario. On the contrary, running a similar node for *temperature* scenario is requires 50% less processing runtime for the computational of *event* stream results; this is due to the smaller size of emitted *events* after they have been preprocessed and matched. Nonetheless, such high computational result per standalone node is due to the fact that deploying more *bolts* requires more processing runtime for each operational function for every module in the DCAD. Thus, an alternative solution is to deploy more computer nodes in the DCAD framework with adding more bolts in to the DAG topology to improve the detection computational runtime performance. For example, increasing the number of nodes from 1 to 8 is evident that the processing runtime of the detection computational is improved by approximately 50%. The performance of DCAD method to computes and process road traffic *event* stream *tuples* over n = 8 nodes is significantly approved from 65 to 45 milliseconds (refer to figure 5.12).

This indicates the effectiveness of DCAD approach, specifically, to reduces the overhead in each computer node regardless scaled up of the *event* stream size.

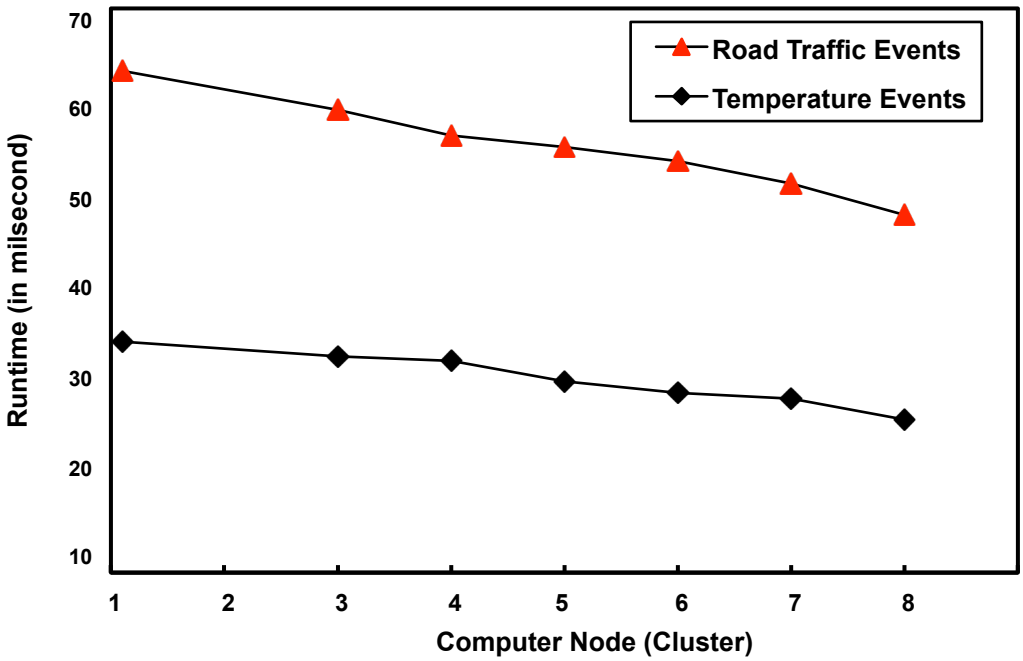


Figure 5.11: Comparison of event streams performance cluster runtime.

5.9. Prediction Error and Performance Measure

To evaluate the performance of the proposed model and algorithms, there are number of evaluation metrics of sequential analysis are suggested in the literature; a) *PH* test is capable to monitor the prediction error and change in the *event* streams according to *AScore* as predefined in between [0,1]. b) for the *Contextual Anomaly* $\boldsymbol{C_A}$ performance result evaluation, the value for threshold is settled according to *AScore* < *t* threshold. In this context, if the probability of value *v* of e_i <0.5, then the computation ratio considered as positive result. On the contrary, if the probability of value *v* of e_i> 0.5, it assumes that the computational ratio is negative; this is due to the increasing the number of

changes in the *event* streams data distribution during the algorithm's learning process.

The learning procedure by CESA is defined based on the result of how competitive the proposed algorithm is in relation to the change detection and prediction of error computational rates. The result of anomalous *event* detection is also evaluated based on scalability performance of the algorithm's runtime process as described in the previous section. Accordingly, the most common and widely proposed evaluation metrics are *Holdout* and *Prequential* to estimate prediction computational error rate (Mouss et al., 2004). The former metric is complex and computationally expensive to test the algorithm over high volumes of *event* streams. For this reason, such metric is irrelevant due to the size of proposed *event* streams and distributed stream structure model. On the other hand, *Prequential* evaluation metric is mainly based on test-and-train procedure, thus, this metric is more reliable than the previous metric to estimate the performance of CESA algorithm prediction error rate. The learning prediction error process is mainly based on computed accumulative sum of the loss function error as denoted $L(f)$ from *n* number of *event* streams. The fading factor parameter is set as $\alpha = 0.5$ for both MAE and RMSE and the average of *Prequential* error computes over window partitions of w = 200,000 *event* stream *tuples*; hence, such values for decay factor is an ideal value to measure the error rates. The total sum of absolute deviations *t* and the number of the values of v_i are used to learn from *n* number of *event* streams. Since new *event* stream e_i arrived for the training process, *n* updates respectively. In relation to monitoring *event* streams statues whether they have been covered by the rule or not, *PH* test is used to compute the loss function error based on MAE or RSME. The lower v_i probability values state unusual change in the *event* stream tuples. This is disregarded or uncovered by the rule-set in the learning process and they have been considered as *Contextual* anomalous *event* according to the *AScore* probability computational rates. The summary description of the CESA algorithm's parameters is demonstrated in Table 5.4.

Allocated Nod	Event Tuples	Executed Latency	Process Latency	PA	CA
n1	250000	0.048	0.04	178300	71700
n2	250000	0.033	0.02	201421	48579
n3	250000	0.047	0.03	224821	25179
n4	250000	0.023	0.02	119565	130435
n5	250000	0.022	0.01	185896	64104
n6	250000	0.029	0.03	223454	26546

Table 5-4: CESA algorithm experimental evaluation parameters.

The result of the prediction error rate to test CESA algorithm performance is primarily based on the both Equation 3.9 and 3.10 as depicts in figure 5.13 and 5.14. The *Prequential* evaluation error demonstrates the effectiveness of $\boldsymbol{C_A}$ model, which is satisfied to the predefined threshold rate to train the model before any changes occurs in the *event* stream. This is achieved by training window partition as w_p = 200,000, length of L = 1,000,000 size of *event* stream tuples. The experiment results have indicated that the rule set computational error of C_A model for each *event* stream is competitive according to the evaluation mean results from MAE and RSME metrics. Such metric is proposed to find if *event* stream is covered by the rule *r* or is disregarded during the training detection as can be seen in Appendix 5.

Figure 5.13 depicts several changes in the *event* streams during the learning by the CESA algorithm. RSME represents in Δ, which indicates the result for the mean square error according to predefined fading factor range, while the result of prediction error slightly decreases from 0.4 to 0.3. Thus, this is indicating a positive result while the size of *event* stream is increasing. This demonstrates the 95% accuracy of the result from testing C_A model for prediction error in the *event* streams. On the other hand, MAE represented in blue (×) symbol and consists of several points which indicates the change in the streams sequence over the time; however, as the size of *event* stream is scaled up, the change rate is decreases due to the less occurred changes in the *event* stream behaviour and window partitioning mechanism. This can demonstrates how accurate the model

predicts the error rates and is able to rapidly adapt to the change from high volumes of *event* stream partitioning in parallel.

In relation to the stability of CESA algorithm, middle dash line in both figures 5.13. and 5.14 and represents how competitive are each *event* stream conditions in both case studies in terms of change. The result is evaluated based on the probability of *AScore* according in CESA to measure whether the result of *AScore* is aligned with a computational range according to Equations 3.7 and 3.8 as < 0.5 (positive) or > 0.5 (negative). Specifically, negative score result is associated with *event* stream with *Contextual* behaviour while positive score refers to anomalous *event* without *Contextual* behaviour. The dash (-) line represents the predefined threshold t <0.25, while the threshold line can be considered the predicted score results cross the line is indicating the normal behaviour of the *event* streams, and the blew the line is considered as anomalous score values.

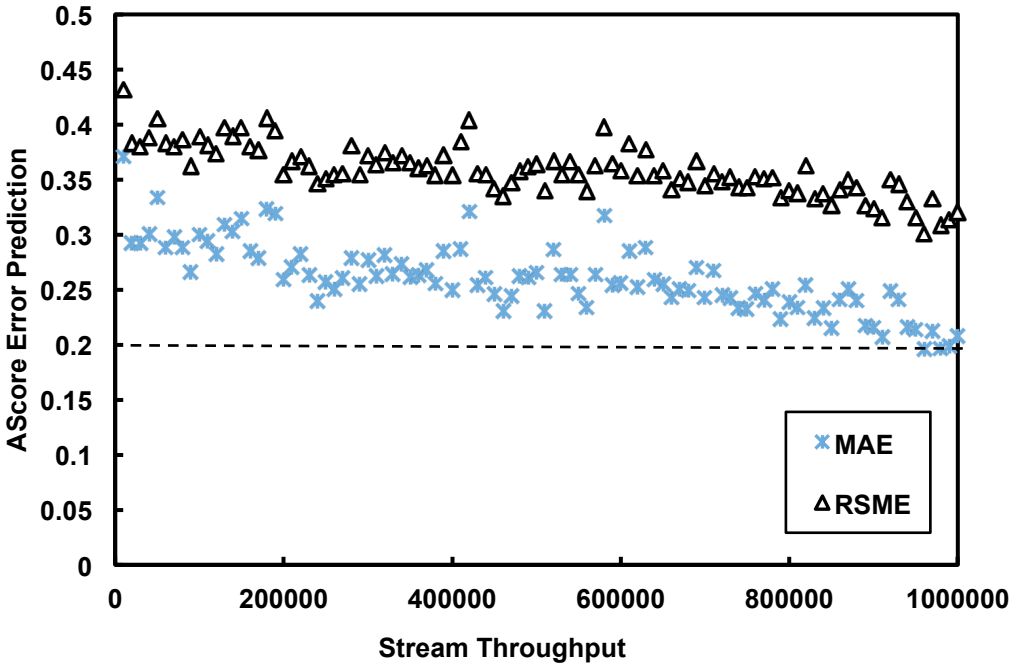


Figure 5.12: CESA algorithm accuracy result by evaluating prequential predicting error metrics (MAE and RSME) using temperature event streams.

In contrast to the pervious case study results, in road traffic scenario both MAE and RSME results have shown to be less competitive due the high number of peaks. Such changes are due to the fact that the average function of the *event* stream error is higher. This indicates that the *event* stream condition is less stable due to the number of changes in the speed *event* streams *tuples*. Such behaviour is indicating that *event* streams evolved over the time of predicting error. The results of MAE in both case scenarios have been more stable. Importantly, such results are indicating the stability of the CESA algorithm in time evolving situation and the model stability in relation to the detecting accuracy scoring rate.

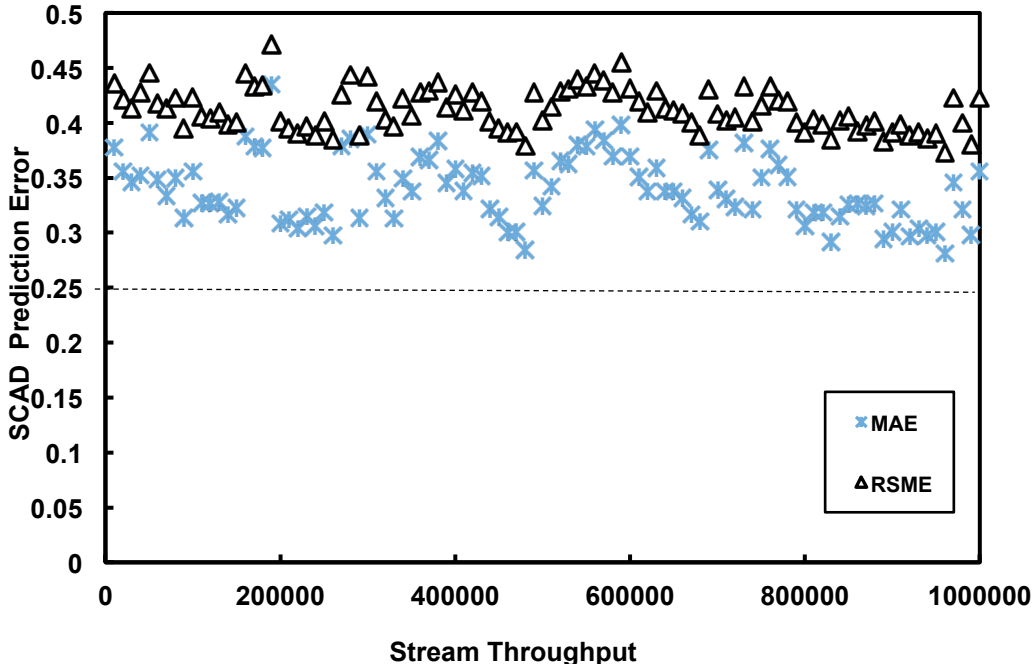


Figure 5.13: CESA algorithm accuracy result by evaluating prequential predicting error metrics (MAE and RSME) using highway road traffic event streams.

Figure 5.14 presents both MAE and RSME results for 10,000 *event* streams per window partitions. The last two columns of the table are referring to the probability result of *event* values (refer to Appendix 5). Hence, the accuracy of both MAE and RMSE depends on the cost function error to measures the

accuracy rate; the low cost corresponds with high accuracy while the lower error result is indicating the high accuracy of the algorithm in relation to change detection prediction. In contrast, high error prediction indicates the instability of the algorithm in terms of delay in recovering after the change is detected.

5.10. Changes Detection Results by the CESA Algorithm

Change detecting metric is relying on the sum of false alarms and true positive rates to measure and estimate how compatible is the algorithm is. In such condition, *event* streams have been tested based on high mean and false alarm rate, which is detected by the CESA algorithm. The main result of change detection depends on setting variant parameters as α and λ in PH test. For the first case study of *temperature*, 100,000 *event* streams have been tested with a range of threshold rate for rule set with change detection as $r > 0 > \alpha$, count threshold $k > 1$ which is $\lambda = 100$ as presented in *y-axis*, and $w_p = 10{,}000$ *event* stream *tuples* per window. The rule number in each window partition increases since the rate of change slightly increases while *event* streams tuple value change over the time as shown in figure 5.15). The rate of false alarm is primarily depending on the size of *event* streams in each w_p, the larger size of w_p is resulting in high rate of changes in the *event* streams.

The size of the *event* stream experimental result throughput of 1,000,000 *tuples* shown in *x-axis* (refer to figure 5.15). Thus, this indicates that smaller size of window partition is resulting of low rate of false alarm and change in the *event* streams. In contrast, larger size of window partition indicates lower computational results.

In relation to highway road traffic case study, change detection is tested over 1 million stream *tuples* as presented in *x-axis* with similar parameter ranges rule set $r > 0 > \alpha$, count threshold $k > 1$ which is $\lambda = 100$, and $w_p = 100{,}000$. In this situation, the size of stream *tuples* per window partition is scaled up from

100,000 to 1,000,000 to train the C_A model for a false alarm rate. In contrast to the pervious scenario, the number of false alarm rate of change in the *event* streams linearly is increased 50% on average as shown in figure 5.15. For example, compare pervious case study with the result of road traffic change rates, the number of false alarm rate changes in the *event* streams from 0 increased to 10 in the first window, 20 to 60 in the last window partitioning. This is due to the larger size of *event* streams in this case study and high number of change rates, which indicates *event* stream *tuple* value, is changing over the time. In this context, an ideal solution is to reduce the risk of high number of false alarm rate by decreasing the value of α =0.001. Importantly, as the value of α is decreased the range of λ is increased. Conceptually, increasing the size of *event* streams per window partition indicates the raise of false alarm rate due to the number changes in the *event* streams *tuple* values.

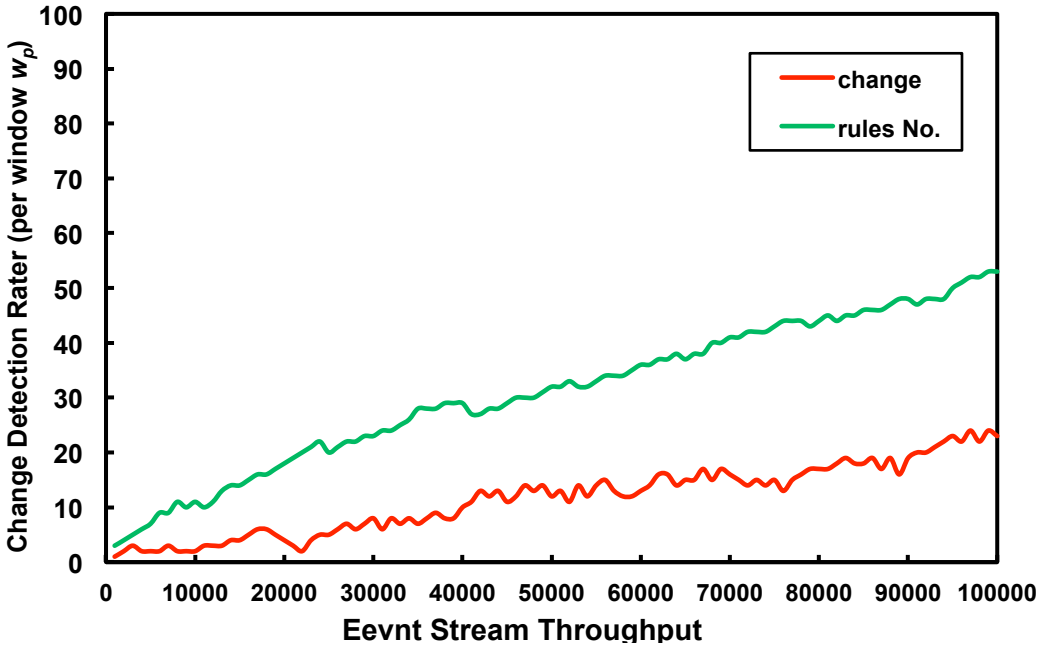


Figure 5.14: Detecting change rates by CESA algorithm for every wp = 10,000.

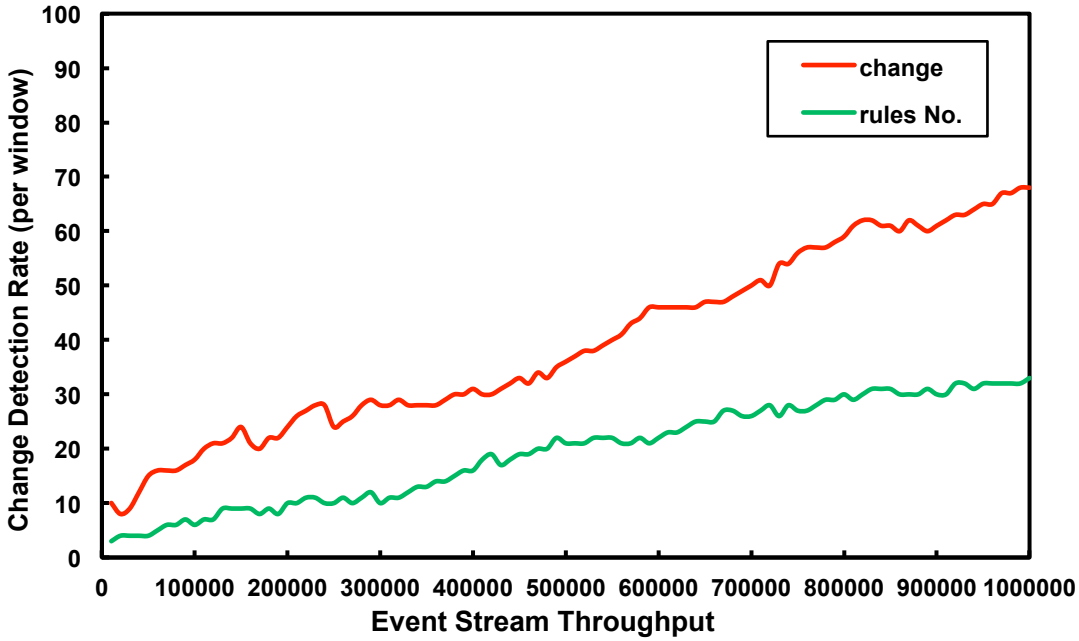


Figure 5.15: Detecting change rates by CESA algorithm for every w_p = 100,000.

Chapter Summary

The experimental results of distributed *event* stream detection have been presented in this chapter. First, the result of aggregation for the pre-processing and *event* streams cleaning is presented before evaluating the result of the algorithms. The result of *event* streams partitioning according to both window models (*count-based, time-based*) is presented for the two case studies. The main aim of such result is to compute several *events* per window partitioned in parallel. Second, the results of detecting *Contextual* anomalous *events* by CESA algorithm have been presented in both IoT case studies. The *Contextual* behaviour was considered based on several facts including the time of the *event* occurrence and *tuple* value in every stream sequence. The performance of CESA algorithm was evaluated based on detecting changes in the *event* streams according to the *PH* test results. The algorithm is evaluated in terms of capability to handle high throughput *event* streams and ability to compute *AScore* according to C_A model. Importantly, the rates of prediction error are evaluated to measure how effective the CESA algorithm is in relation to detecting changes in the *event* a stream. Third, the performance of DCAD framework has been tested over a distributed cluster based on processing time, low-latency of each module's framework through a variety of experiments. The performance of DCAD is evaluated based on high *event* streams throughput, *n* number of window partitioning that had a major impact on the processing time and response. Importantly, the evaluation of parallelism is described based on different facts such as scaling up the size of *event* streams, increasing the number of computer nodes, however, the number of windows partition is evaluated to estimate the processing time.

Chapter 6

Conclusion and Future Work

This chapter summarises the main research contributions of this thesis with overall the empirical result discussion, followed by future works future research extend opportunities.

6.1. Conclusion

Due to the advances in the digital technology, the size of generating data stream is scaling up very rapidly. Such concept has motivated to study this research by investigating the research problems in the core research field of data stream mining and machine learning to understand the concept of online learning from data streams, its model and stream constraints (e.g., *time*, *space* and *accuracy*). To discover anomalous *event* from sensor streams with such constraints, first, we defined the stream processing data structure model; second, we designed C_A model to define the *Contextual* behaviour of the *event* sensor streams. In this context, distributed stream processing and detecting unusual *events* including *Contextual* anomalous detection behaviour from real time application is playing an important role. Specifically, detecting high volumes of anomalous *events* from IoT sensor stream requires a robust method and novel algorithms to design and to handle high rate of unbounded sequence of *event* streams real time. Importantly, since sensor stream data attributes and values are correlated, to

best our knowledge detecting *Contextual* method is disregards in the previous research studies for the following justified reasons,

a) In batch (static) learning, anomaly is mainly detected through multi-scan learning process; for example, the dataset can be divided into two tasks of test and train learning, in contrast to streaming data, when in online learning process; space, time, and accuracy are major concerns to be considered, which is due to the size and rate of the data streams. However, detecting anomalous *event* in online learning is one of the most common appropriate techniques in most of the real time applications when intelligent decision making is playing an important role.

b) Detecting *Contextual* behaviour from dynamic *event* streams in online learning can be proposed to predict unusual *events* from the data resources. However, such concept is disregarded in the most of streaming applications, which is due to the dynamic change in the data stream during the data distribution.

c) In relation to the scalability of streaming data, specifically, for the big IoT data, most of the traditional anomaly detection methods are mainly proposing data reduction to overcome the scalability concern rather than distributed mining. However, such method is only capable for a limited size of the data and it is inappropriate for high volumes of data streams. Thus, in recent years, distributed data stream processing is proposed as an alternative solution.

Generally,, anomalous *events* are occur in the real time and the size of the data stream is very large. Thus, we first proposed novel window partitioning methods according to the learning tasks of count and time-based from arriving *event* streams, and *event* streams then divided into several partitions to handle the high rate of streams. The main benefit of this approach is to handle *event* streams data distribution and detect or control unusual changes to the streams.

Second, we designed a novel distributed *Contextual event* stream detection method to detect the *Contextual* behaviour of the sensor data in real time based on the stream data structure model. A detailed description of the main research contributions is summarized as follows.

Event stream anomaly detection: a theoretical study of the anomaly detection background is described in Chapter two with the concept of anomaly detection in different research disciplines such as statistical analysis, data mining, stream mining, and machine learning. In addition to this, several common detection methods such as supervised, semi-supervised and unsupervised learning are studied to identify research limitations. On the one hand, we defined and designed each concept of *event* streams, such as *Contextual Anomaly* $\boldsymbol{C_A}$ model based on distributed *event* stream data structure model. The $\boldsymbol{C_A}$ model is designed to define the *Contextual* behaviour according to the possibility of assigning *AScore* to the designed partitioned of *event* streams. In this context, CESA algorithm is designed to detect the *Contextual* behaviours after the anomalous *event* streams are partitioned. The algorithm is designed to first check the *event* streams status as they continuously arrive from match module in the DCAD, then *event* streams have been trained by the ***CA*** model to check whether *event* stream tuple values are associating to with the stored matching rule set. If the *event* streams are not matched, then such *event* streams are considered as *Contextual* anomalous *event* or *event* streams alternatively removed from the sequence list in each window partition.

Distributed *Contextual* Anomalous Event: to deal with the stream scalability concern, we proposed a novel solution to handle a high rate of sequence of the even streams. Specifically, we designed a novel distributed method so-called DCAD as shown in Appendix 2, and the main task of this method is to able to process and detect high volumes of *Contextual event* streams. DCAD consists of three distributed computational modules; Pre-Processing, Event matching, and *Contextual* Detection. These models are primarily based on distributed stream processing data structure model. The first module is designed to pre-

processing the *event* streams, only *events* with similar tuple values are considered for further processing and detection since matching module is associated with matching the *event* stream tuples and handling the rate of *event* streams to capture any change that possibly occurred during the *event* stream data distribution. Finally, we implemented CESA algorithm in the *Contextual* module to detect *Contextual* behaviour from the *event* streams based on their *AScore* computational stream output result values per each window partitions.
The experimental results demonstrated the effectiveness of each fact which we have measured and tested based on the on the *event* stream size, the number of window partitions, and scaling up the processing time. The result shows that distributed processing performance is more efficient than centralised approach to compute and detect high number of *Contextual event* behaviour in real time. The main drawback of centralised computation is the number of designed computational functions, which we have used per each bolt in the topology to perform by the workers in parallel. The topology is required to deploy two spouts and ten bolts together to perform such computational functions per only one standalone node. Second, the performance of DCAD framework is evaluated based on increased and decreased of a number window partitions; the experimental result is demonstrating that using more windows per computer node can have a major impact on the performance of the processing time.

Importantly, using less number of windows per computer node with a smaller size of window partition length demonstrated a significant improvement in terms of using less memory space, computational runtime process, and detecting *event* streams changes. For such reasons, window partitions are proposed across the DCAD framework and the experimental results indicated the effectiveness of DCAD. The detecting experimental results are evaluated based on scaling up the *event* streams size to test the capability of each computing node based on their processing runtime performance. The experimental result is satisfied the assumption of detection scalability concern of over one million *event* streams per less than one second. An alternative

solution is to deploy more nodes in the DCAD framework with adding more bolts in the DAG topology for each case study. For example, adding from 1 to 8 nodes could decrease the processing time by approximately 50% in parallel. Consider DCAD framework over 8 nodes, the result shows a significant improvement in the performance from 65 to 45 milliseconds to process and detect road traffic *event* streams. This can be an ideal approach to reduce overhead in each computer node as the size of *event* streams have been scaled up. The DCAD approach can be extended and implemented for other problems within the other real world application domains including for credit card fraud detection, network security monitoring system, weather prediction, and medical sensor monitoring to detect anomalous *events* over the data streams in real time.

6.2. Future Work

In this thesis, we aimed to fulfill the main thesis's objectives of detecting anomaly from high volumes of stream, specifically, addressing the problem of distributed *Contextual* anomalous *event* stream. The proposed solution and results can be extended in further studies as described in below.

i. New window modeling-based method can easily be deployed in application like social media stream data by allocating Anomaly Score *AScore* to k^{th} nearest window partition and train $\boldsymbol{C_A}$ model based on new arriving data w_s where the partition size of each window can possibly have an impact on processing of the computational result. *Contextual* snapshot model can be designed based on matching dissimilar collection of data according to their context and time-series behaviour. This can be achieved by dividing arriving *event* streams into different snapshot time-based interval window partitions. The anomaly snapshot model can be built on collections of dissimilar *Contextual* behaviour. Thesis problem can be further investigated in unsupervised learning; for example, *Clustream* Clustering algorithm can be proposed to groups similar *event*

streams according to their tuple values in each window partition. Gaussian predictor, α and β are the most common appropriate coefficients metrics to detect any changes within the *event* streams. The *event* streams can be set into three metric intervals of t = 8, t = 16, t = 24 per each window partitioning length of 10,000 stream tuples.

ii. Future investigation can be considered to detect anomalous *event* streams from other application domains with multi-variant data attributes and high dimensional data streams; for example, grouping and partitioning *event* attribute values according to their *Contextual* attribute such as segment of *events* per window partition. Prior to the problem of anomaly detection and *event* stream partitioning and parallel detection. It is worth to study and investigate concept drift detection in other time-series applications.

iii. Offline and online distributed anomaly detection approach is another future research direction to be studied. This approach is already proposed in some of the research disciplines of data mining and machine learning; thus, we believed that distributed hybrid anomaly detection research is an interesting and challenging task to be studied for the future work. This can be achieved by; first, building anomaly model from historical *event* data behaviour in offline, and training the arriving new *event* streams on online from new window partitions.

iv. In relation to the anomaly scalability drawback, we have proposed distributed cluster of computer nodes, while alternative solution can be implementing a cloud-based architecture. Cloud architecture is one of an efficient appropriate approach to aggregate and process high volumes of stream detection. One of a key problem in distributed messaging system is a communication channel between the servers. For example, when a topology deployed on Storm nimbus, it is necessary to have a communication channel destination to define tasks between the DCAD modules and broker. To address such problem, we designed

communication channel between the master-servers architecture by connecting Kafka brokers with Storm framework through apache Zookeeper [16]. This is another remaining research investigation of dynamic distribution of *event* between the distributed modules and monitoring the data distribution behaviours.

[16] https://zookeeper.apache.org/

Bibliography

Abadi, D. J., Carney, D., Etintemel, U., Cherniack, M., Convey, C., Lee, S., IDaster-servers architecture by connecting Kafka brokers with Storm framework through*The VLDB Journal The International Journal on Very Large Data Bases*, *12*(2), 120 139. https://doi.org/10.1007/s00778-003-0095-z

Abadi, M., Barham, P., Chen, J., Chen, Z., Davis, A., Dean, J., es Lee, S., IDaster-servers architecture by connecting Kafka brokers with Storm framework through apache Zookeeper detection. proce*12th USENIX Symposium on Operating Systems Design and Implementation (OSDI '16)* (pp. 265IX Symposium on Operating Systems Desi

Aggarwal, C. C., & Wang, J. (2007). Data Streams: Models and Algorithms. *Data Streams*, *31*, 9a Streams C., & Wang, J. (2007). Data Streams:

Aggarwal, C. (2016). *Outlier Analysis*. New York: Yorktown Heights.

Aggarwal, C. C. (2012). A segment-based framework for modelling and mining data streams. *Knowledge and Information Systems*, *30*(1), 1dge and Information Systems-based framework fo

Aggarwal, C. C., Watson, T. J., Ctr, R., Han, J., Wang, J., & Yu, P. S. (2003). A Framework for Clustering Evolving Data Streams. *Proc. of the 29th Int. Conf. on Very Large Data Bases*, 81. of the 29th Int. Conf. on Very La

Agneeswaran, V. S. (2014). *Big Data Analytics Beyond Hadoop*. *Big Data Analytics Beyond Hadoop*. Retrieved from http://ptgmedia.pearsoncmg.com/images/9780133837940/samplepages/0133837947.pdf

Akcora, C. G., Carminati, B., Ferrari, E., & Kantarcioglu, M. (2014). Detecting anomalies in social network data consumption. *Social Network Analysis and Mining*, *4*(1), 1 Network Analysis and Miningrari, E., & Kantar

Akter, S., & Wamba, S. F. (2016). Big data analytics in E-commerce: a systematic review and agenda for future research. *Electronic Markets*, *26*(2), 173–194. https://doi.org/10.1007/s12525-016-0219-0

Alam, A., & Ahmed, J. (2014). Hadoop architecture and its issues. In *Proceedings - 2014 International Conference on Computational Science and Computational Intelligence, CSCI 2014* (Vol. 2, pp. 2884 Internatis://doi.org/10.1109/CSCI.2014.140

Almeida, E., Ferreira, C., & Gama, J. (2013). Adaptive model rules from data streams. In *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)* (Vol. 8188 LNAI, pp. 480 Science (including subseries Lecture Notes in Arti

Amini, A., Saboohi, H., & Wah, T. Y. (2013). A multi density-based clustering algorithm for data stream with noise. In *Proceedings - IEEE 13th International Conference on Data Mining Workshops, ICDMW 2013* (pp. 1105s - IEEE 13th International Conference on Data

Amini, A., Wah, T. Y., & Saboohi, H. (2014). On density-based data streams clustering algorithms: A survey. *Journal of Computer Science and Technology*. https://doi.org/10.1007/s11390-014-1416-y

Andrade, H., Gedik, B., Wu, K.-L., & Yu, P. S. (2011). Processing high data rate streams in System S. *Journal of Parallel and Distributed Computing*, *71*(2), 145of Parallel and Distributed Computing. (2011). P

Angiulli, F., & Fassetti, F. (2010). Distance-based outlier queries in data streams: The novel task and algorithms. *Data Mining and Knowledge Discovery*, *20*(2), 290ing

anhttps://doi.org/10.1007/s10618-009-0159-9

Arasu, A., Babcock, B., Babu, S., Cieslewicz, J., Ito, K., Motwani, R., … Widom, J. (2004). STREAM: The Stanford Data Stream Management System. *Concrete*, (2004e., Babcock, B., Babu, S., Cieslewicz, J., Ito, K., Motwani, R., … Widom, J.

Amen, B., & Lu, J., (2015). Sketch of Big Data Real time Analytics Model. *The Fifth International Conference on Advances in Information Mining and Management (IMMM)*, 21st - 26th June 2015, Brussels, Belgium.

Assunt - 26th June 2015, Brussels, Belgium.ces in Information Mining and Management (IMMM). STREAM: The Stanford Data Stream Management Syctions. *Journal of Parallel and Distributed Computing*, *79–80*, 3rnal of Parallel and Distributed Computings in

Atzori, L., Iera, A., & Morabito, G. (2010). The Internet of Things: A survey. *Computer Networks*, *54*(15), 2787etworkshings: A survey. G. (2010).omnet.2010.05.010

Babcock, B., Babu, S., Datar, M., Motwani, R., & Widom, J. (2002). Models and issues in data stream systems. In *Proceedings of the twenty-first ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systemssystems. In* (p. 1). https://doi.org/10.1145/543613.543615

Badarna, M., & Wolff, R. (2014). Fast and accurate detection of changes in data streams. *Statistical Analysis and Data Mining*, *7*(2), 125cal Analysis and Data Miningt and accur

Bai, M., Wang, X., Xin, J., & Wang, G. (2015). An efficient algorithm for distributed density-based outlier detection on big data. *Neurocomputing*, *181*, 19ocomputing X., Xin, J., & Wang, G. (2015). An eff

Baldoni, R., Querzoni, L., Tarkoma, S., & Virgillito, A. (2009). Distributed event routing in publish/subscribe systems. In *Middleware for Network Eccentric and Mobile Applications* (pp. 219e for Network Eccentric and Mobile ApplicationsIn 0

Beigi, M. S., Chang, S.-F., Ebadollahi, S., & Verma, D. C. (2011). Anomaly detection in information streams without prior domain knowledge. *IBM Journal of Research and Development*, *55*(5), 11:1-11:11. https://doi.org/10.1147/JRD.2011.2163280

Bhatnagar, V., Kaur, S., & Chakravarthy, S. (2014). Clustering data streams using grid-based synopsis. *Knowledge and Information Systems*, *41*(1), 127e and Information Systemsvarthy, S. (2014). Clu

Bifet, A., Morales-bueno, R., Baena-Garcia, M., Campo-Avila, J. Del, Fidalgo, R., Bifet, A., synopsis. lier detection on big data. ment Syctions. framewor*4th ECML PKDD International Workshop on Knowledge Discovery from Data Streams* (Vol. 6, pp. 77ternational Workshop on Knowledge Di

Bifet, A., de Francisci Morales, G., Read, J., Holmes, G., & Pfahringer, B. (2015). Efficient Online Evaluation of Big Data Stream Classifiers. *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, (April 2016), 59 21th ACM SIGKDD International Conference on

Bifet, A. (2009). Adaptive Learning and Mining for Data Streams and Frequent Patterns. *Dissertation Universitat Politecnica de Catalunya*, *11*(1), 55ation Universitat Politecnica de Catalunyar

Bondu, A., & Boullrsitat PolitecA supervised approach for change detection in data streams. *International Joint Conference on Neural Networks (IJCNN)*, 8. Retrieved from http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=6033265%0Ahttp://www.marc-boulle.fr/publications/BonduEtAlIJCNN11.pdf

Brzezinski, D., & Stefanowski, J. (2014). Reacting to different types of concept drift: The accuracy updated ensemble algorithm. *IEEE Transactions on Neural Networks and Learning Systems*, *25*(1), 81ansactions on Neural Networks and Learning Syst

Cao, F., Ester, M., Qian, W., & Zhou, A. (2006). Density-based clustering over an evolving data stream with noise. *Proceedings of the Sixth SIAM International Conference on Data Mining*, *2006*, 328edings of the Sixth SIAM International Confer

Caron, E., & De Assuncao, M. D. (2017). Multi-criteria malleable task management for hybrid-cloud platforms. In *Proceedings of 2016 International Conference on Cloud Computing Technologies and Applications, CloudTech 2016* (pp. 326gs of 2016 International Conference on Cloud Computin

Chakrabarti, K., Keogh, E., Mehrotra, S., & Pazzani, M. (2002). Locally adaptive dimensionality reduction for indexing large time series databases. *ACM Transactions on Database Systems*, *27*(2), 188sactions on Database Systems, S., & Pazzani

Candela, V., Banerjee, A., & Kumar, V. (2012). Anomaly detection for discrete sequences: A survey. *IEEE Transactions on Knowledge and Data Engineering*. https://doi.org/10.1109/TKDE.2010.235

Candela, V., Banerjee, A., & Kumar, V. (2009). Anomaly detection: A survey. *ACM Computing Surveys (CSUR)*, *41*(September), 1Survehttps://doi.org/10.1145/1541880.1541882

Chandrasekaran, S., Cooper, O., Deshpande, A., Franklin, M. J., Hellerstein, J. M., Hong, W., y reduction for indexing large time series databases. amework through apache Zookeeper *Cidr*, *20*(March), 668. https://doi.org/10.1145/872757.872857

Chaudhry, N., Shaw, K., & Abdelguerfi, M. (2005). *Stream Data Management* (1st ed.). US: Springer US

Chen, M., Mao, S., & Liu, Y. (2014). Big data: A survey. In *Mobile Networks and Applications* (Vol. 19, pp. 171d Applications14). Big/10.1007/s11036-013-0489-0

Cugola, G., & Margara, A. (2012). Processing flows of information. *ACM Computing Surveys*, *44*(3), 1mputing Surveys, A. (2012). Processing flows

De Matteis, T., & Mencagli, G. (2016). Keep calm and react with foresight: Strategies for Low- Latency and Energy-Efficient Elastic Data Stream Processing. *Proceedings of the 21st ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming - PPoPP d E*, 1ceedings of the 21st ACM SIGPLAN Symposium o

Demceedings of the 21st ACM S8). Detecting concept drift in data streams using model explanation. *Expert Systems with Applications*, *92*, 546t Systems with Applications Detecting concept dr

Ding, S., Wu, F., Qian, J., Jia, H., & Jin, F. (2015). Research on data stream clustering algorithms. *Artificial Intelligence Review*, *43*(4), 593al Intelligence Reviewa, H., & Jin, F. (2015).

Dobre, C., & Xhafa, F. (2014). Parallel programming paradigms and frameworks in Big Data Era. *International Journal of Parallel Programming*, *42*(5), 710ional Jours://doi.org/10.1007/s10766-013-0272-7

Doulkeridis, C., & N/doi.org/10.1007/s10766-013-0272-7adigms and frameworks in Big Data Era. rithms. PP *VLDB Journal*. https://doi.org/10.1007/s00778-013-0319-9

Department for Business, Energy & Industrial Strategy. (2016). Smart Meters Quarterly Report to End September: final report. Retrieved from https://www.gov.uk/government/uploads/system/uploads/attachment_data/file/579197/2016_Q3_Smart_Meters_Report_Final.pdf

Duarte, J., Gama, J., & Bifet, A. (2016). Adaptive Model Rules From High-Speed Data Streams. *ACM Transactions on Knowledge Discovery from Data*, *10*(3), 1ansactions on Knowledge Discovery fr

Erfani, S. M., Rajasegarar, S., Karunasekera, S., & Leckie, C. (2016). High-dimensional and

large-scale anomaly detection using a linear one-class SVM with deep learning. *Pattern Recognition*, *58*, 121rn Recognition detection using a linear one-class

Esposito, C., Ficco, M., Palmieri, F., & Castiglione, A. (2015). A knowledge-based platform for big data analytics based on publish/subscribe services and stream processing. *Knowledge-Based Systems*, *79*, 3wledge-Based Systems, Palmieri, F., & Castiglione

Eugster, P. T., Felber, P. A., Guerraoui, R., & Kermarrec, A.-M. (2003). The many faces of publish/subscribe. *ACM Computing Surveys*, *35*(2), 114uting Surveyser, P. A., Guerraoui, R., & Ke

Facca, F. M., & Lanzi, P. L. (2005). Mining interesting knowledge from weblogs: A survey. *Data and Knowledge Engineering*. https://doi.org/10.1016/j.datak.2004.08.001

Fahad, A., Alshatri, N., Tari, Z., Alamri, A., Khalil, I., Zomaya, A. Y., … Bouras, A. (2014). A survey of clustering algorithms for big data: Taxonomy and empirical analysis. *IEEE Transactions on Emerging Topics in Computing*, *2*(3), 267nsactions on Emerging Topics in Computinghalil9

Fan, J., & Liu, H. (2013). Statistical analysis of big data on pharmacogenomics. *Advanced Drug Delivery Reviews*. https://doi.org/10.1016/j.addr.2013.04.008

Faria, E. R., Gon/10.1016/j.addr.2013.04.008sis of big data on pharmacogenomics. as, A. (2014). A ction in data streams. *Artificial Intelligence Review*, *45*(2), 235al Intelligence Reviewdr.2013.04.008sis of big

Faria, E. R., Gama, J., & Carvalho, A. C. (2013). Novelty detection algorithm for data streams multi-class problems. *Proceedings of the 28th Annual ACM Symposium on Applied Computing*, 795edings of the 28th Annual ACM Symposium on Ap

Farid, D. M., Zhang, L., Hossain, A., Rahman, C. M., Strachan, R., Sexton, G., & Dahal, K. (2013). An adaptive ensemble classifier for mining concept drifting data streams. *Expert Systems with Applications*, *40*(15), 5895tems with Applications, A., Rahman, C. M., Strach

Ferrer-Troyano, F., Aguilar-Ruiz, J. S., & Riquelme, J. C. (2005). Incremental rule learning based on example nearness from numerical data streams. In *Proceedings of the 2005 ACM symposium on Applied computing - SAC '05* (p. 568). https://doi.org/10.1145/1066677.1066808

Folino, G., & Sabatino, P. (2016). Ensemble based collaborative and distributed intrusion detection systems: A survey. *Journal of Network and Computer Applications*. https://doi.org/10.1016/j.jnca.2016.03.011

Gama, J., Medas, P., Castillo, G., & Rodrigues, P. (2004). Learning with drift detection. *Brazilian Symposium on Artificial Intelligence*, 286lian Symposium on Artificial Intelligences,45-5_29

Gama, J. (2012). A survey on learning from data streams: current and future trends. *Prog Artif Intell*, *1*, 45 Artif IntellA survey on learning from data st

Gama, J., SebastiA survey on learning from data streams: current and future trends. ction. delgorithms. In *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining - KDD n s* (p. 329). https://doi.org/10.1145/1557019.1557060

Gama, J., https://doi.org/10.1145/1557019.1557060onference on Knowledge discovery rvey on concept drift adaptation. *ACM Computing Surveys*, *46*(4), 1mputing Surveysi.org/10.1145/1557019

Gao, X., Ferrara, E., & Qiu, J. (2015). Parallel clustering of high-dimensional social media data streams. In *Proceedings - 2015 IEEE/ACM 15th International Symposium on Cluster, Cloud, and Grid Computing, CCGrid 2015* (pp. 323gs - 2015 IEEE/ACM 15th International Symposi

Garc 323gs - 2015 IEEE/ACM 15th International Symposium on Cluster, Cloud, and Grid Computing, CCGrid 2015In ation. y. rom numTechniques, systems and challenges. *Computers & Security*, *28*, 18uters & SecurityEEE/ACM 15th International Symp

Genesereth, M., Keller, A., & Duschka, O. (1997). Infomaster: an information integration system. *SIGMOD Rec*, 26(2), 539-542.

Golmohammadi, K., & Zaiane, O. R. (2015). Time series contextual anomaly detection for detecting market manipulation in stock market. In *Proceedings of the 2015 IEEE International Conference on Data Science and Advanced Analytics, DSAA 2015*. https://doi.org/10.1109/DSAA.2015.7344856

Grosse, V., & Turin, F. (2012). Stream mining: A novel architecture for ensemble-based classification. *Knowledge and Information Systems*, *30*(2), 247e and Information Systemsel architecture for en

Gupta, M., Gao, J., Aggarwal, C. C., & Han, J. (2014). Outlier Detection for Temporal Data: A Survey. *Knowledge and Data Engineering, IEEE Transactions on*, *26*(9), 2250 and Data Engineering, IEEE Transactions on4

Hashem, I. A. T., Yaqoob, I., Anuar, N. B., Mokhtar, S., Gani, A., & Ullah Khan, S. (2015). The rise of "big data" on cloud computing: Review and open research issues. *Information Systems*, *47*, 98rmation SystemsYaqoob, I., Anuar, N. B., Mokht

Hayes, M. A., & Capretz, M. A. (2015). Contextual anomaly detection framework for big sensor data. *Journal of Big Data*, *2*(1). https://doi.org/10.1186/s40537-014-0011-y

Hoens, T. R., Polikar, R., & Chawla, N. V. (2012). Learning from streaming data with concept drift and imbalance: an overview. *Progress in Artificial Intelligence*, *1*(1), 89s in Artips://doi.org/10.1007/s13748-011-0008-0

Hu, H., Wen, Y., Chua, T. S., & Li, X. (2014). Toward scalable systems for big data analytics: A technology tutorial. *IEEE Access*, *2*, 652Accessn, Y., Chua, T. S., & Li, X. (2014). Toward

Huang, H., & Kasiviswanathan, S. P. (2015). Streaming anomaly detection using randomized matrix sketching. *Proceedings of the VLDB Endowment*, *9*(3), 192ngs of the VLDB EndowmentP. (2015). Streaming

Ikonomovska, E., Gama, J., & DntP. (2015). Streaming anomaly detection using randomizeata streams. *Data Mining and Knowledge Discovery*, *23*(1), 128ing and Knowledge Discovery (2015). Streaming a

IEEE Society, I.-S. E. (2007). IEEE Standard for Standard General Requirements for Liquid-Immersed Distribution, Power, and Regulating Transformers

Jiang, Y., Zeng, C., Xu, J., & Li, T. (2014). Real time contextual collective anomaly detection over multiple data streams. *Workshop on Outlier Detection & Description under Data Diversity (ODD)*, *1*, 23shop on Outlier Detection & Description unde

Kamburugamuve, S., Christiansen, L., & Fox, G. (2015). A framework for real time processing of sensor data in the cloud. *Journal of Sensors*, *2015*. https://doi.org/10.1155/2015/468047

Karatepe, I. A. & Zeydan, E. (2014). Anomaly Detection In Cellular Network Data Using Big Data Analytics. *European Wireless 2014; 20th European Wireless Conference; Proceedings of*.

Karunaratne, P., Karunasekera, S., & Harwood, A. (2017). Distributed stream clustering using micro-clusters on Apache Storm. *Journal of Parallel and Distributed Computing*, *108*, 74nal of Parallel and Distributed Computing, A. (

Katal, A., Wazid, M., & Goudar, R. H. (2013). Big data: Issues, challenges, tools and Good practices. In *2013 Sixth International Conference on Contemporary Computing ({IC}3)*

(pp. 404–409). https://doi.org/10.1109/IC3.2013.6612229

Kifer, D., Ben-david, S., & Gehrke, J. (2004). Detecting Change in Data Streams. In *the 30th International Conference on Very Large Data Bases Conference* (pp. 180International Conference on Very Large71(94)90421-9

Kmieciak, M. R., & Stefanowski, J. (2011). Handling Sudden Concept Drift in Enron Messages Data Stream. *Control and Cybernetics*, 667ol an

Krawczyk, B., Minku, L. L., Gama, J., Stefanowski, J., & WoConcept Drift in Enron Messages Data Streama stream analysis: A survey. *Information Fusion*, *37*, 132mation Fusionku, L. L., Gama, J., Stefanowski, J.,

Krempl, G., Spiliopoulou, M., Stefanowski, J., ski, J., & WoConcept Drift in Enron Messages Data Sievi, S. (2014). Open challenges for data stream mining research. *ACM SIGKDD Explorations Newsletter*, *16*(1), 1GKDD Explorations Newsletteranowski, J., ski

Kreps, J., Narkhede, N., & Rao, J. (2011). Kafka: a Distributed Messaging System for Log Processing. *ACM SIGMOD Workshop on Networking Meets Databases*, 6. Retrieved from http://research.microsoft.com/en-us/um/people/srikanth/netdb11/netdb11papers/netdb11-final12.pdf

Kuncheva, L. I. (2008). Classifier ensembles for detecting concept change in streaming data: Overview and perspectives. *Proceedings of the Second Workshop SUEMA, ECAI 2008*, (July), 5 of the Second Workshop SUEMA, ECAI 2008etecting con

Lee, C. H., & Chien, T. F. (2013). Leveraging microblogging big data with a modified density-based clustering approach for event awareness and topic ranking. *Journal of Information Science*, *39*(4), 523of Information Scienceata with a modified dens

Li, G., & Jacobsen, H.-A. (2005). Composite subscriptions in content-based publish/subscribe systems. In *Middleware a05: Proceedings of the ACM/IFIP/USENIX 2005 International Conference on Middleware* (pp. 249e a05: Proceedings of the ACM/IFIP/USEN

Li, H.-F., & Lee, S.-Y. (2009). Mining frequent itemsets over data streams using efficient window sliding techniques. *Expert Systems with Applications*, *36*(2), 1466stems with ApplicationsMining frequent itemsets o

Liu, Y., & Plale, B. (2003). Survey of publish subscribe event systems. *Indiana University Department of Computer Science*, (TR574), 1ersity Department of Computer Scienx.ist.psu.edu/viewdoc/download?doi=10.1.1.67.3753&rep=rep1&type=pdf

Ma, J., & Perkins, S. (2003). Time-series novelty detection using one-class support vector machines. *Proceedings of the International Joint Conference on Neural Networks, 2003.*, *3*, 1741–1745. https://doi.org/10.1109/IJCNN.2003.1223670

Ma, J., Sun, L., Wang, H., Zhang, Y., & Aickelin, U. (2016). Supervised Anomaly Detection in Uncertain Pseudoperiodic Data Streams. *ACM Transactions on Internet Technology*, *16*(1), 1ansactions on Intern/10.1145/2806890

Ma, Y., Wu, H., Wang, L., Huang, B., Ranjan, R., Zomaya, A., & Jie, W. (2015). Remote sensing big data computing: Challenges and opportunities. *Future Generation Computer Systems*, *51*, 47re Generation Computer Systems., Ranjan, R., Zoma

Mahapatra, A., Srivastava, N., & Srivastava, J. (2012). Contextual anomaly detection in text data. *Algorithms*, *5*(4), 469ms anomaly detection in text data. , J

McCreadie, R., Macdonald, C., Ounis, I., Osborne, M., & Petrovic, S. (2013). Scalable distributed event detection for Twitter. In *Proceedings - 2013 IEEE International Conference on Big Data, Big Data 2013* (pp. 543gs - 2013 IEEE International Conference on Big Data

Mirsky, Y., Shabtai, A., Shapira, B., Elovici, Y., & Rokach, L. (2017). Anomaly detection for

smartphone data streams. *Pervasive and Mobile Computing*, *35*, 83asive and Mobile Computingra, B., Elovici, Y., &

Mouss, H., Mouss, D., Mouss, N., & Sefouhi, L. (2004). Test of page-hinckley, an approach for fault detection in an agro-alimentary production system. *Proceedings of the 5th Asian Control Conference*, 815edings of the 5th Asian Control Conference(200

Muthukrishnan, S. (2005). Data Streams: Algorithms and Applications. *Foundations and Trends5). Data Streams: Algorithms and* , *1*(2), 117ons and Tre://doi.org/10.1561/0400000002

Nguyen, H.-L., Woon, Y.-K., & Ng, W.-K. (2014). A survey on data stream clustering and classification. *Knowledge and Information Systems*, *45*(3), 535e and Information Systems W.-K. (2014). A surve

OReilly, C., Gluhak, A., Imran, M., & Rajasegarar, S. (2014). Anomaly Detection in Wireless Sensor Networks in a Non-Stationary Environment. *Ieeexplore.Ieee.Org*, *16*(3), 1lore.Ieee.Org, A., Imran, M., & Rajasegarar, S. (20

Papadimitriou, S., Sun, J., & Faloutsos, C. (2005). Streaming pattern discovery in multiple time-series. *Eedings of the 31st International Conference on Very Large Data Bases*, 697gs of the 31st International Conference on Very Large Data Bas

Parthasarathy, S., Ghoting, a, & Otey, M. (2007). *A survey of distributed mining of data streams*. *Data Streams*. https://doi.org/10.1007/978-0-387-47534-9_13

Pham, D., Venkatesh, S., Lazarescu, M., & Budhaditya, S. (2012). Anomaly detection in high volume data stream networks. *Data Mining and Knowledge Discovery*. 28(1), 145-189.

Philip Chen, C. L., & Zhang, C.-Y. (2014). Data-intensive applications, challenges, techniques and technologies: A survey on Big Data. *Information Sciences*, *275*, 314mation Sciences& Zhang, C.-Y. (2014). Data-inte

Rathore, M. M., Ahmad, A., & Paul, A. (2016). Real time intrusion detection system for ultra-high-speed big data environments. *Journal of Supercomputing*, *72*(9), 3489f Supercomputingtion system for ultra-high-speed

Rauber, T., & Rrcomputingtion sys*Parallel programming: For multicore and cluster systems*. *Parallel Programming: For Multicore and Cluster Systems*. https://doi.org/10.1007/978-3-642-37801-0

Rettig, L., Khayati, M., Cudre-Mauroux, P., & Piorkowski, M. (2015). Online anomaly detection over Big Data streams. In *2015 IEEE International Conference on Big Data (Big Data)* (pp. 1113International Conference on Big Data (Big Data) M. (

Saleh, O., Hagedorn, S., & Sattler, K.-U. (2015). Complex Event Processing on Linked Stream Data. *Datenbank-Spektrum*, *15*(2), 119k-Spektrumn, S., & Sattle1007/s13222-015-0190-5

Schneider, M., Ertel, W., & Ramos, F. (2016). Expected similarity estimation for large-scale batch and streaming anomaly detection. *Machine Learning*, *105*(3), 305Learningrtel, W., & Ramos, F. (2016). Expected

Silva, J. A., Faria, E. R., Barros, R. C., Hruschka, E. R., Carvalho, A. C. P. L. F. de, & Gama, J. (2013). Data stream clustering. *ACM Computing Surveys*, *46*(1), 1mputing SurveysE. R., Barros, R. C., Hruschk

Solaimani, M., Iftekhar, M., Khan, L., & Thuraisingham, B. (2014). Statistical technique for online anomaly detection using Spark over heterogeneous data from multi-source VMware performance data. In *Proceedings - 2014 IEEE International Conference on Big Data, IEEE Big Data 2014* (pp. 1086s - 2014 IEEE International Conferencea.2014.7004343

Su, L., Han, W., Zou, P., Jia, Y., & Yang, S. (2007). Continuous Kernel-Based Outlier

Detection over Distributed Data Streams. *Frontiers of High Performance Computing and Networking ISPA 2007 Workshops*, 305iers of High Performance Computing -540-74767-3_32

Tao, Y., & Ozsu, M. (2009). Mining data streams with periodically changing distributions. *Proceedings of the 18th ACM conference on Information and knowledge management*. Hong Kong, China, ACM: 887-896.

Tanbeer, S. K., Ahmed, C. F., Jeong, B. S., & Lee, Y. K. (2009). Sliding window-based frequent pattern mining over data streams. *Information Sciences*, *179*(22), 3843n Sciencesd, C. F., Jeong, B. S., & Lee, Y. K. (

Tatbul, N. (2010). Streaming data integration: Challenges and opportunities. *ICDE Workshops*, 155Workshops10). Streaming data integration: Challe

Tran, D.-H., Gaber, M. M., & Sattler, K.-U. (2014). Change Detection in Streaming Data in the Era of Big Data: Models and Issues. *ACM SIGKDD Explorations Newsletter - Special Issue on Big Data*, (1), 30D Explorations Newsletter - Special Issue on

Tsai, C.-W., Lai, C.-F., Chao, H.-C., & Vasilakos, A. V. (2015). Big data analytics: a survey. *Journal of Big Data*, *2*(1), 21. https://doi.org/10.1186/s40537-015-0030-3

Vallim, R. M. M., & De Mello, R. F. (2014). Proposal of a new stability concept to detect changes in unsupervised data streams. *Expert Systems with Applications*, *41*(16), 7350tems with Applicationsept to detect changes in un

Van Vlasselaer, V., Bravo, C., Caelen, O., Eliassi-Rad, T., Akoglu, L., Snoeck, M., & Baesens, B. (2015). APATE: A novel approach for automated credit card transaction fraud detection using network-based extensions. *Decision Support Systems*, *75*, 38sion Support Systemso, C., Caelen, O., Eliassi

Vu, A. T., De Francisci Morales, G., Gama, J., & Bifet, A. (2014). Distributed Adaptive Model Rules for mining big data streams. In *2014 IEEE International Conference on Big Data (Big Data)* (pp. 345 International Conference on Big Data (Big Data). (

Wang, X.-T., D.-R. Shen, M. Bai, T.-Z. Nie, Y. Kou and G. Yu (2015). An Efficient Algorithm for Distributed Outlier Detection in Large Multi-Dimensional Datasets. *Computer Science and Technology*, 30(6), 1233-1248.

Wu, S., & Wang, S. (2013). Information-theoretic outlier detection for large-scale categorical data. *IEEE Transactions on Knowledge and Data Engineering*, *25*(3), 589nsactions on Knowledge and Data Engineering

Yan, Y., J. Zhang, B. Huang, X. Sun, J. Mu, Z. Zhang and T. Moscibroda (2015). Distributed Outlier Detection using Compressive Sensing. *In Proceedings of the 2015ACM SIGMOD International Conference on Management of Data,* (pp. 3-16).

Xie, M., Han, S., Tian, B., & Parvin, S. (2011). Anomaly detection in wireless sensor networks: A survey. *Journal of Network and Computer Applications*, *34*(4), 1302f Network and Computer Applications11). Anomaly d

Yang, H., & Fong, S. (2015). Countering the concept-drift problems in big data by an incrementally optimized stream mining model. *Journal of Systems and Software*, *102*, 158al of Systems and.org/10.1016/j.jss.2014.07.010

Ye, R., & Li, X. (2017). Collective Representation for Abnormal Event Detection. *Journal of Computer Science and Technology*, *32*(3), 470of Computer Science and Technologyentation for

Young, W. C., Blumenstock, J. E., Fox, E. B., & Mccormick, T. H. (2014). Detecting and classifying anomalous behaviour in spatiotemporal network data. *Kdd-Lesi*.

Yu, L., & Lan, Z. (2016). A Scalable, Non-Parametric Method for Detecting Performance Anomaly in Large Scale Computing. *IEEE Transactions on Parallel and Distributed*

Systems, *27*(7), 1902sactions on Parallel and Distributed SystemsMeth

Zhang, J. (2013). Advancements of Outlier Detection: A Survey. *ICST Transactions on Scalable Information Systems*, *13*(1), e2. https://doi.org/10.4108/trans.sis.2013.01-03.e2

Zhang, X., Fang, Z., Wen, Y., Li, Z. & Qiao, Y. (2016). Range Loss for Deep Face Recognition with Long-tail: *CoRR ArXiv e-prints*, http://dblp.org/rec/bib/journals/corr/ZhangFWL016

Zhang, J., Gao, Q., & Wang, H. (2008). SPOT: A system for detecting projected outliers from high-dimensional data streams. In *Proceedings - International Conference on Data Engineering* (pp. 1628s - International Conference on Data Engineeringd

Zhang, Meratnia, N., & Havinga, P. (2010). Outlier Detection Techniques for Wireless Sensor Networks: A Survey. *IEEE Communications Surveys & Tutorials*, *12*(2), 159munications Surveys & Tutorials0). Outlier Detection

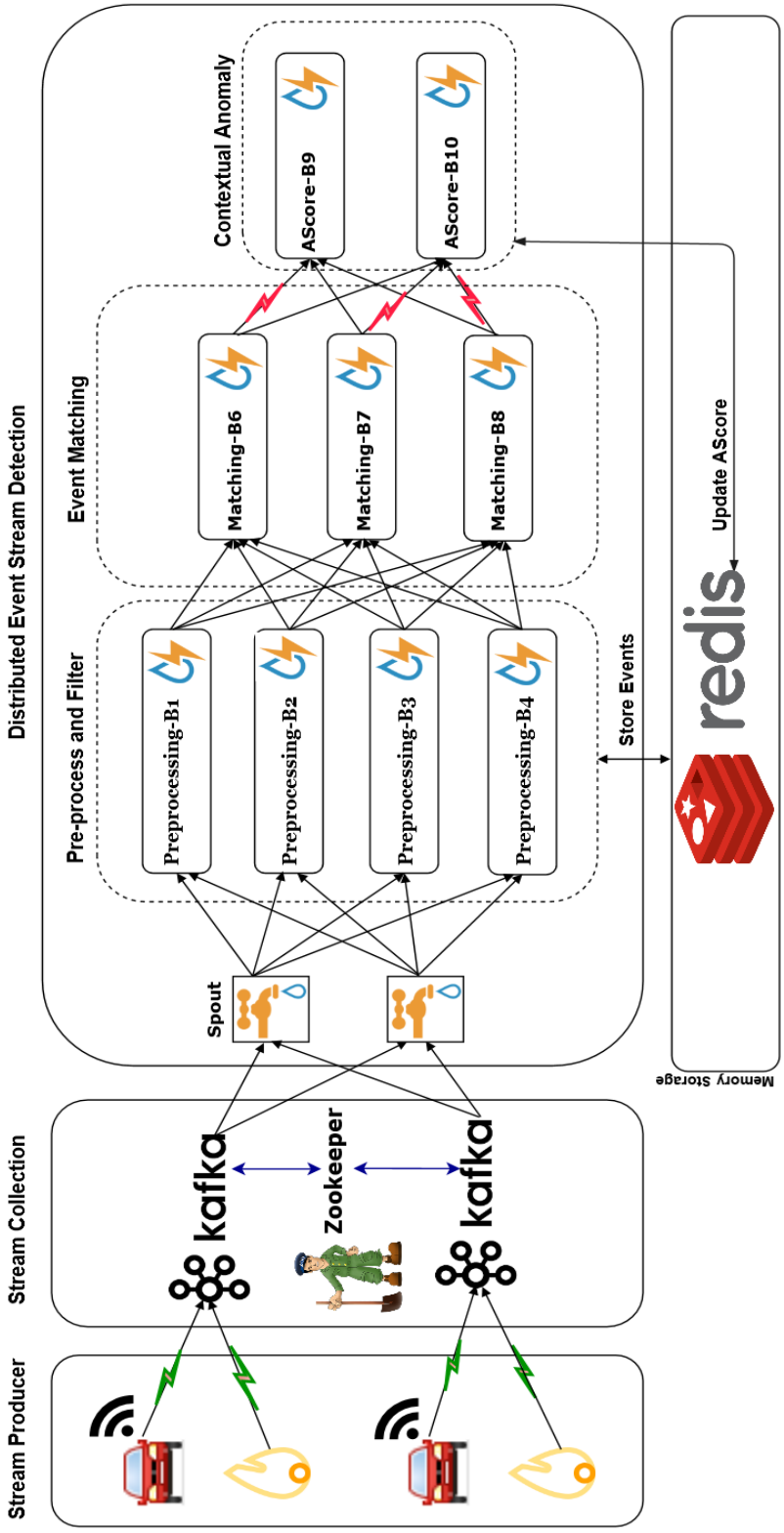
Zhang, T., Ramakrishnan, R., & Livny, M. (1997). BIRCH: A New Data Clustering Algorithm and Its Applications. *Data Mining and Knowledge Discovery*, *1*(2), 141ing and Knowledge Discoveryy, M. (1997). BIRC

Zheng, Y., Zhang, H., & Yu, Y. (2015). Detecting collective anomalies from multiple spatio-temporal datasets across different domains. In *Proceedings of the 23rd SIGSPATIAL International Conference on Advances in Geographic Information Systems - GIS oss* (pp. 1ings of the 23rd SIGSPATIAL International Con

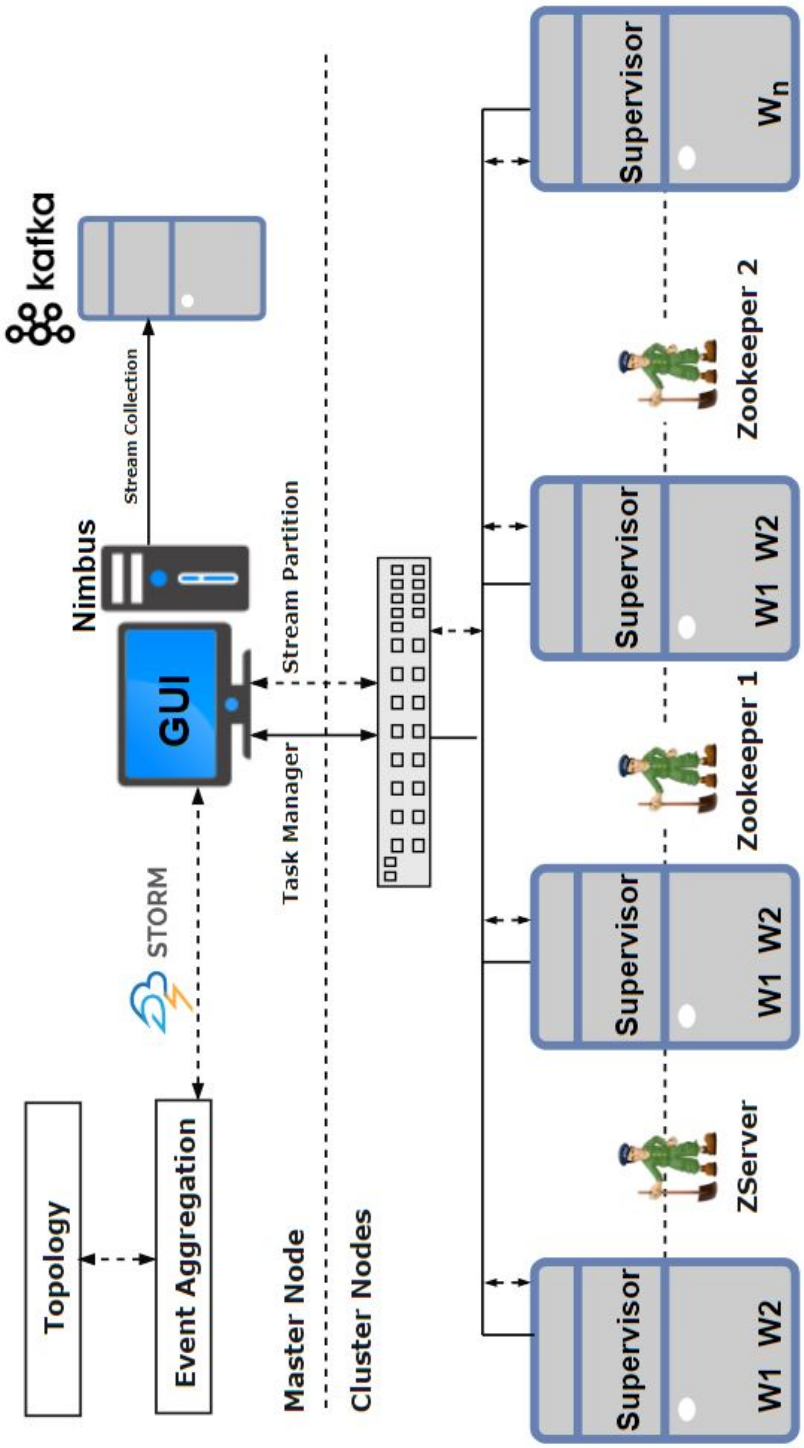
Appendix 1: Big Data State-of-the-art Comparison

Big Data Streaming State-of-the-art						
Technology – Developed	Architecture	Programming Model	Structure Type	Advantages	Data Handling	References
Storm - 2011 by Twitter	Master-Slave	Directed Acyclic Graph	Un-structured Real time Streaming process	Embeddable networking library API Scalable Fault-tolerant Reliable	In Memory	(Hu, Wen et al., 2014, Philip Chen, 2014)
S4 – 2010 by Yahoo	Master-Slave	Processing Elements (PEs)	Un-structured Real time Streaming processing	Distributed, Scalable, Fault-tolerant Pluggable platform Easy develop applications	Buffer read Memory	(Dobre, 2014, Hu, Wen et al., 2014, Philip Chen, 2014) (Hu, Wen et al., 2014)(Hu, Wen et al., 2014)
Spark – 2014 by Berkeley AMPLab	Hybrid Application	Resilient Distributed Dataset (RDD)	Combine SQL, Streaming, Complex analytics	More Reliable than Hadoop Fast in-memory Scalability Fault tolerance	In-memory computing, Different operations (join, sort, filter).	(Alam, 2014); (Dobre, 2014)
Kafka - 2011 by LinkedIn	Publish-Subscribe Messaging System	*Topic*-based	Real time streaming process Un-structured	High-throughput stream of unchallengeable activity Data	Message Queue Memory	(Marcos D. Assunção a , 2014, Philip Chen, 2014)

Appendix 2: Distributed Contextual Anomaly Detection (DACD) Framework



Appendix 3: Distributed Contextual Anomaly Detection (DCAD) Architecture



Appendix 4: Result of Event Stream Window Partitions

Stream *Tuple*	Temp <20℃	Temp >26℃	Stream *Tuple*	Temp <20℃	Temp >26℃
10000	1	3	**510000**	42	19
20000	3	5	**520000**	45	17
30000	2	6	**530000**	45	19
40000	1	7	**540000**	46	19
50000	3	7	**550000**	47	19
60000	3	8	**560000**	48	18
70000	3	7	**570000**	48	19
80000	4	8	**580000**	49	20
90000	4	10	**590000**	49	21
100000	6	9	**600000**	50	21
110000	8	8	**610000**	52	20
120000	8	9	**620000**	53	21
130000	8	10	**630000**	54	22
140000	10	8	**640000**	56	21
150000	12	7	**650000**	57	21
160000	12	9	**660000**	58	21
170000	12	10	**670000**	58	22
180000	14	9	**680000**	59	22
190000	15	10	**690000**	60	22
200000	16	10	**700000**	60	23
210000	16	11	**710000**	60	24
220000	18	11	**720000**	61	23
230000	18	12	**730000**	61	24
240000	18	13	**740000**	63	23
250000	19	13	**750000**	63	24
260000	20	14	**760000**	64	24
270000	23	13	**770000**	64	25
280000	25	12	**780000**	66	24
290000	25	12	**790000**	66	24
300000	26	12	**800000**	67	24
310000	27	12	**810000**	68	25
320000	28	12	**820000**	69	25
330000	28	13	**830000**	70	26
340000	29	14	**840000**	71	26
350000	31	14	**850000**	73	25
360000	31	15	**860000**	73	26
370000	32	15	**870000**	75	26
380000	32	16	**880000**	78	24
390000	32	17	**890000**	78	25
400000	32	18	**900000**	78	26
410000	33	17	**910000**	79	26
420000	35	17	**920000**	81	25
430000	35	19	**930000**	84	24
440000	35	20	**940000**	85	24
450000	37	18	**950000**	86	25
460000	37	19	**960000**	86	26
470000	37	20	**970000**	87	26
480000	38	19	**980000**	88	26
490000	39	19	**990000**	88	28
500000	41	19	**1000000**	89	28

Appendix 5: Result of MAE and RSME Predicating Error by CESA

Stream *Tuples*	MAE	RSME	Stream *Tuples*	MAE	RSME
10000	0.377734	0.436237	**510000**	0.342148	0.414309
20000	0.355729	0.421506	**520000**	0.365707	0.428173
30000	0.346158	0.412526	**530000**	0.36283	0.43084
40000	0.352114	0.427611	**540000**	0.380327	0.439482
50000	0.391087	0.445245	**550000**	0.378281	0.433275
60000	0.348688	0.417421	**560000**	0.393992	0.444702
70000	0.333416	0.412919	**570000**	0.384196	0.43875
80000	0.349678	0.422812	**580000**	0.36961	0.427712
90000	0.313396	0.394406	**590000**	0.398304	0.454473
100000	0.355356	0.423262	**600000**	0.36985	0.431453
110000	0.327224	0.405802	**610000**	0.3509	0.419499
120000	0.326414	0.404108	**620000**	0.337364	0.409793
130000	0.328005	0.409912	**630000**	0.359118	0.429203
140000	0.316664	0.39871	**640000**	0.337389	0.412642
150000	0.322265	0.400518	**650000**	0.337508	0.410686
160000	0.387403	0.444668	**660000**	0.331523	0.408405
170000	0.37841	0.432911	**670000**	0.316325	0.400751
180000	0.376901	0.433851	**680000**	0.310247	0.388417
190000	0.434962	0.470736	**690000**	0.375616	0.430434
200000	0.30836	0.400807	**700000**	0.338798	0.407799
210000	0.312019	0.395385	**710000**	0.330764	0.402369
220000	0.303664	0.390776	**720000**	0.323569	0.404949
230000	0.314901	0.397049	**730000**	0.381713	0.432922
240000	0.305793	0.389099	**740000**	0.321164	0.401144
250000	0.318161	0.401564	**750000**	0.350293	0.415828
260000	0.297087	0.384984	**760000**	0.375969	0.432984
270000	0.379114	0.426261	**770000**	0.361567	0.420211
280000	0.385466	0.443657	**780000**	0.350948	0.419361
290000	0.313561	0.388028	**790000**	0.321395	0.400226
300000	0.389672	0.441858	**800000**	0.30614	0.390959
310000	0.355884	0.419721	**810000**	0.318342	0.402854
320000	0.331611	0.403205	**820000**	0.318409	0.398578
330000	0.313279	0.397064	**830000**	0.291221	0.38457
340000	0.349326	0.421857	**840000**	0.31505	0.402135
350000	0.337274	0.406027	**850000**	0.32583	0.405458
360000	0.369044	0.427574	**860000**	0.326691	0.392481
370000	0.365935	0.429176	**870000**	0.324392	0.397487
380000	0.383801	0.43677	**880000**	0.326381	0.40105
390000	0.345053	0.41386	**890000**	0.294519	0.383192
400000	0.357952	0.425975	**900000**	0.30102	0.391097
410000	0.338022	0.411441	**910000**	0.321264	0.398287
420000	0.354474	0.427024	**920000**	0.297009	0.388732
430000	0.351719	0.419726	**930000**	0.304015	0.390988
440000	0.321922	0.401809	**940000**	0.297213	0.385752
450000	0.314427	0.395105	**950000**	0.301051	0.38969
460000	0.300295	0.391369	**960000**	0.281209	0.37304
470000	0.300824	0.391583	**970000**	0.345609	0.422641
480000	0.284679	0.379111	**980000**	0.321314	0.400043
490000	0.357031	0.427721	**990000**	0.298108	0.380865
500000	0.323929	0.402367	**1000000**	0.35585	0.422783

Appendix 6: Result of CESA Computational CPU (in Millisecond)

Stream *Tuples*	Evaluation time (CPU in ms)	MAE	RSME	Stream *Tuples*	Evaluation time (CPU in ms)	MAE	RSME
10000	0.764405	0.377734	0.436237	**510000**	39.81146	0.27199	0.370555
20000	1.419609	0.318078	0.398038	**520000**	40.63826	0.286167	0.381862
30000	2.074813	0.317481	0.396094	**530000**	41.49627	0.276589	0.373995
40000	2.761218	0.317787	0.402433	**540000**	42.36987	0.308565	0.397129
50000	3.494422	0.374653	0.432863	**550000**	43.21228	0.339709	0.416241
60000	4.196427	0.319512	0.400157	**560000**	44.07028	0.33858	0.416693
70000	4.914032	0.306783	0.392703	**570000**	44.92829	0.335415	0.412187
80000	5.631636	0.310344	0.400669	**580000**	45.78629	0.339409	0.419439
90000	6.364841	0.322444	0.40104	**590000**	46.6287	0.307729	0.39508
100000	7.098046	0.313482	0.400602	**600000**	47.5179	0.313478	0.395766
110000	7.76885	0.323636	0.405941	**610000**	48.40711	0.321348	0.40493
120000	8.502055	0.303892	0.393001	**620000**	49.29632	0.304444	0.393414
130000	9.266459	0.327823	0.404883	**630000**	50.20112	0.319027	0.402461
140000	9.968464	0.319278	0.403377	**640000**	51.16833	0.313392	0.395451
150000	10.73287	0.312568	0.399848	**650000**	52.07313	0.327152	0.405836
160000	11.45047	0.319046	0.399314	**660000**	53.00914	0.320973	0.403861
170000	12.18368	0.294529	0.388603	**670000**	54.00755	0.33089	0.410473
180000	12.94808	0.300365	0.39198	**680000**	54.92795	0.304707	0.393328
190000	13.69689	0.292	0.383244	**690000**	55.91076	0.281573	0.375513
200000	14.44569	0.28163	0.377077	**700000**	56.86236	0.274878	0.367752
210000	15.2257	0.314629	0.397056	**710000**	57.81397	0.255975	0.358194
220000	16.0057	0.278633	0.374838	**720000**	58.73438	0.261346	0.362881
230000	16.78571	0.285248	0.367978	**730000**	59.68598	0.270085	0.36789
240000	17.55011	0.252158	0.352088	**740000**	60.63759	0.260343	0.355274
250000	18.34572	0.343128	0.411512	**750000**	61.57359	0.256542	0.357336
260000	19.12572	0.293045	0.376176	**760000**	62.494	0.266119	0.362714
270000	19.92133	0.282907	0.376096	**770000**	63.43001	0.265823	0.37081
280000	20.70133	0.299676	0.39582	**780000**	64.45961	0.312816	0.399184
290000	21.51254	0.294843	0.38106	**790000**	65.41122	0.297663	0.386887
300000	22.27694	0.329247	0.402235	**800000**	66.40963	0.253794	0.354401
310000	23.10375	0.307977	0.392931	**810000**	67.36123	0.271014	0.373055
320000	23.88375	0.282988	0.377551	**820000**	68.31284	0.296194	0.3832
330000	24.69496	0.277338	0.372752	**830000**	69.28004	0.246577	0.350968
340000	25.53736	0.288326	0.381658	**840000**	70.26285	0.270313	0.36863
350000	26.42657	0.305149	0.389769	**850000**	71.23006	0.267249	0.37038
360000	27.28457	0.287235	0.374947	**860000**	72.22846	0.255626	0.355875
370000	28.08018	0.286994	0.374396	**870000**	73.21127	0.288104	0.380478
380000	28.92259	0.284692	0.38201	**880000**	74.22528	0.275211	0.369883
390000	29.76499	0.274043	0.37009	**890000**	75.19248	0.251714	0.356947
400000	30.6074	0.275243	0.374456	**900000**	76.20649	0.269389	0.369061
410000	31.4654	0.277032	0.373196	**910000**	77.2048949	0.28758	0.383283
420000	32.27661	0.268367	0.363379	**920000**	78.2501016	0.25027	0.355447
430000	33.10341	0.272262	0.370499	**930000**	79.2953083	0.268328	0.366343
440000	33.93022	0.274671	0.373451	**940000**	80.3249149	0.265356	0.369605
450000	34.74142	0.301644	0.386576	**950000**	81.3077212	0.266455	0.367765
460000	35.56823	0.282809	0.375297	**960000**	82.2905275	0.258483	0.354583
470000	36.39503	0.301681	0.388574	**970000**	83.2577337	0.260006	0.360383
480000	37.23744	0.284054	0.377967	**980000**	84.2561401	0.266881	0.362767
490000	38.09544	0.285035	0.379343	**990000**	85.2545465	0.253947	0.353746
500000	38.93785	0.272656	0.37418	**1000000**	86.2529529	0.270685	0.370323